オブジェクト指向型モデル記述と
その定性シミュレーションに関する研究

平成２年12月

堂園　浩

目次

第1章 序論

近年，プラントに代表されるような大規模システムの発展により，その故障の解析の方法の確立が重要な課題となっている．[1] ここでいう故障の解析とは，ある故障が発生した場合に，システムにどのような症状が発生するか，また逆に，システムにある症状が見られるときに，どのような故障が発生しているかを解析することである．前者の意味での故障解析は故障シミュレーション，後者は故障診断と呼ばれる．また，故障シミュレーションは，その結果から故障と症状の対応表を作成することで，故障診断にも応用することができる．特に，大規模なシステムを対象とする場合は，診断に用いる知識ベース（診断ベース）が膨大で専門家自身の手による直接の構築は困難であり，計算機による支援が必要となる．このような方法の１つとしては，故障シミュレーションによる診断ベース作成が有効である．

本研究では，故障シミュレーションを行うシステムについて考察を行った．また，故障のみでなく，もっと広い意味でシステム内に発生する一般的な事象のシミュレーションについても解析の対象とする．つまり，ある事象がどのように伝播されるのかを解析することになり，このようなシミュレーションは事象伝達シミュレーションと呼ぶ．

事象伝達シミュレーションを扱うデータの形式で分類すると，定量的な方法と定性的な方法に分けられる．定量的な方法では，データとしては実数値データを用い，対象を微分方程式などの厳密な数学的モデルで表現し，その解を求めることで，シミュレーションを行う．[2] ここでいう解は，単純なシステムに対しては解析解が得られることもあるが，ほとんどの場合は数値解として得られる．定性的な方法は，知識工学，人工知能などの応用であり，モデルの記述法，データの形式にも様々なものがあるが，一般的に言えば，記号的，言語的に記述されたデータとモデルを用いて，なんらかの推論を進めて行くことでシミュレーションを進めて行く．[3]

これらの２つの方法を比較すると，定量的な方法では，システムの様々な内部状態

（パラーメータ）に対して，正確な数値解を得ることができるが，その反面，大規模なシステムに対しては状態変数が増大し，シミュレーションに膨大な計算量を必要とする．また，プラントなどには厳密な数学的モデルが得られない部分も存在し，数値的なシミュレーションが不可能な場合もある．定性的な方法では，用いるデータの形式に合わせて得られる結果も曖昧性を含んだものになるが，数値解よりは単純であり，人間には理解し易いものになる．また，モデル自身も曖昧性を持つため，厳密なモデルが得られない場合でもシミュレーションを行うことができる．さらに，システムのパラメータにより厳密に決定されるために，定量的なシミュレーションにおいては得られにくい解を見いだせる場合もある．逆に，定性的なシミュレーションでは，その曖昧性により，様々な種類の解が得られるため，実際には起こり得ない解が得られたり，システムのパラメータによる解の制限が生かされない場合もある．計算量については，得られる解の形式にもよるが，定量的なシミュレーションに比べると少なくてすむ場合が多い．以上に述べたように定性的な方法と定量的な方法では，それぞれ利点，欠点があるが，対象が大規模になると定性的な方法がモデル作成の簡単さ，計算量の点で優れており，また，定性的な方法しか用いられない場合もある．本論文では定性的なシミュレーションに関して，定性モデル構築の方法およびそのシミュレーションアルゴリズムについて研究を行ったものである．

定性モデルには，グラフ構造を用いたもの[4]，定性微分方程式によるもの[5]，フレーム表現によるもの[6]など様々なものが存在する．定性シミュレーションの手法も，モデルの形式に合わせてデバイス中心の方法[4][5]ととプロセス中心の方法[7]に分けられる．デバイス中心の方法では，システムの各々のデバイスに対してその内部および他のデバイスとの相互作用により，シミュレーションが進められ， デバイス自身の記述にはグラフモデルおよび定性方程式がよく用いられる．プロセス中心の方法では，システムの内部で発生するプロセス間の相互作用によりシミュレーションが進められ，プロセスの記述にはフレーム表現，ルール表現などがよく用いられる．いずれにせよ，対象が大規模化すると，このような定性モデルを作成するにあたって何らかの系統的な手段が必要

になる．つまり，デバイス中心にせよプロセス中心にせよ，ある単位ごとにモデルを記述し，それらを統合することで全体のモデルを構築できる枠組みを用意しなければならない．本研究では，そのためにオブジェクト指向アプローチによるモデル化手法[8]を導入する方法について考察を行った．

オブジェクト指向の考え方は各応用分野により若干異なるが，共通するものとしてオブジェクトという概念がある．オブジェクトとは，あるデータとそれを使用するための基本的な操作とは，データと手続きをひとまとめにし，その合併したものを１つのモジュールとみなしたものである．従って，構成しようとするシステムをオブジェクトの集まりとして表現しようという考え方がオブジェクト指向の基本となる．シミュレーションの分野では，対象の記述において，オブジェクト指向の考え方は有用で多くのシステムで用いられている．このことは，1967年にシミュレーション言語として発表されたSimula[9]がオブジェクト指向の考え方の元になっていることや，オブジェクト指向言語の代表的な言語であるSmalltalk[10]が数々のシミュレーションに用いられていることからもわかる．本研究では，定性モデルの構築のために従来のオブジェクト指向アプローチの拡張を行った．

また，このような事象伝達シミュレーションを，対象の記述法の観点から見ると，手続き的な方法と宣言的な方法が考えられる．手続き的な方法とは，FORTRAN等の一般的なプログラミング言語や，GPSS[2]等の従来用いられてきたシミュレーション言語を用い，処理の順序にあわせてプログラムを作成する方法である．宣言的な方法[17]とは，なんらかの知識工学的な記述法を用いた因果関係を記述の集まりで，表現する方法である．例をあげると，「タンクのレベルが３０センチを越えたならば，出力のバルブは閉じられる．」というようなルールの集まりとして全体のシステムを記述する．

手続き的手法では事象に関する知識に加えて，処理のコントロールまでプログラマが作成する必要があるのに対し，宣言的手法では処理のコントロールはシステムが自動的に行うため，事象に関する知識のみ用意することでシミュレーションが行える．そのため，宣言的な方法では，知識獲得からすぐに実行することができ，また知識の追加や削

除も手続き的手法より簡単に行える．本研究では，宣言的な記述法を用いるものとし，具体的には次に述べるような方法で定性モデルを構築する．

定性モデルにおいて，オブジェクトはその内部状態を表す変数と，変数間およびオブジェクト間の関係を表す知識から構成される．本研究では，その知識の記述の方法として，プロダクションルール[11]を用いた方法と，定性ネットワークモデルを用いた方法の２つについて検討した．

プロダクションルールによる方法は，前述の分類ではプロセス中心のモデル化にあたるもので，if-then形式でオブジェクト内で発生する事象間の因果関係を記述し，プロダクションルールの連鎖によりシミュレーションを進めて行く．ただし，対象が時間を含むシステムであるため，プロダクションルールにも時間に関する記述が必要がある．また，実際のシステムでは複数の事象が並行に進むため，ルールの並列実行が行えなければならない．今回は，それらの点に関してプロダクションシステムの拡張を行った．このシステムでは，モデル化の対象は時間情報を必要とするもので，かつ，ルールを用いて記述できるものであり，物理系から，コンピュータネットワークなどまでかなり広範囲なものとなる．また，ルールベースシステムでは，RETEアルゴリズム[12]を用いて条件部のマッチング処理の効率を上げているものが多数存在するが，時間に関する条件のマッチング処理のためにRETEアルゴリズムの改良を行った．

定性ネットワークモデルは，前述の分類ではデバイス中心の考え方であり，変数間，およびオブジェクト間の関係を属性付きのアークで表し，定性ネットワーク上での値伝播によりシミュレーションを進める．ただし，定性ネットワークでは，モデル化の対象は物理系のみに限定される．定性ネットワーク上のシミュレーションにおいては，実行される値伝播が実際の物理変化として起こり得ること，また，定性ネットワーク上で変数間の一貫性が保たれていることが重要となる．本研究では，定性ネットワーク上でそれらの条件を調べながら，シミュレーションを実行するアルゴリズムについて考察を行った．

本論文は次のような５章の構成からなる．１章はこの序論である．２章は時間プロダ

クションシステムによる定性シミュレーションシステムの実現についてである．まず，従来のプロダクションシステムであるOPS5を紹介し，次に時間プロダクションシステムを定義する．その後，マッチング処理のためにRETEアルゴリズムを改良した時間RETEアルゴリズム，およびオブジェクト指向アプローチの導入について述べる．最後にOPS5のRETEアルゴリズムとの比較を行い結言に至る．３章，４章は２章のプロダクションシステムに対し，物理系にのみ対象を絞り，モデル構築に改良を行った定性ネットワークによるシステムの実現について述べる．まず，３章はオブジェクト指向アプローチの拡張による定性ネットワークモデルの記述言語についてである．最初に定性ネットワークモデルを定義し，次にオブジェクト指向アプローチに基づく記述言語の概要について述べる．そして，定性ネットワークモデルの記述のためにオブジェクト指向アプローチの改良である多重継承の拡張とオブジェクトの結合の導入とそれらの実現法について述べ，結言に至る．４章は定性ネットワーク上での，定性シミュレーションアルゴリズムについてである．まず，単純な前向きの伝播のみによる定性シミュレーションの方法について述べる．次に定性ネットワーク上での定性値の一貫性を保ちながら伝播を行うアルゴリズムについて述べ，シミュレーションの例題を挙げ，結言に至る．５章で全体をまとめた結論をのべる．

第2章　時間プロダクションシステムによるシステム構築

２．１　緒言

本章では，定性シミュレーションの１つの方法として，従来のプロダクションシステムに対し離散時間の記述を可能にした離散時間プロダクションシステムとその実行における時間RETEアルゴリズム，およびその知識表現におけるオブジェクト指向アプローチの導入について述べる．

プロダクションシステムとは，自分自身の状態（ワーキングメモリー）をif-thenルール（プロダクションルール）の実行により書き換えることで，推論を進めて行くシステムである．プロダクションシステムの代表的なものとしてはOPS5[11]，OPS83[15]等があり，それらを用いた数多くのアプリケーションが作成されている．本章で扱う離散時間プロダクション・システムとは，一般的なプロダクション・システムの条件部と実行部に離散時間に関する記述を可能にし，動的に変化するシステムを記述したり，またその挙動をシミュレーションしたりするものである．このシステムを用いた応用範囲としては動的システムの事象伝達シミュレーション[16]などが考えられる．具体的に例を挙げると，プラントなどのフェイルセイフシステムやコンピュータネットワークにおける通信プロトコルの検証，また，生態系のシミュレーションなどが考えられる．

しかし，OPS5に代表される従来のプロダクション・システムでは，ルール内に直接には時間を表現できない．また時間を制御するルールを加えて，時間を記述することは可能であるが，この場合はルールの記述性に問題があり，さらに時間を制御するルールを付加することに伴う処理のオーバーヘッドも大きくなる．そこで，プロダクションルールの条件部と実行部に時間に関する記述を加えることで，時間を陽に記述できる離散時間プロダクション・システムを作成した．これまでに，このようなシステムとしては，OPS5のルール実行部のみに時間の記述を可能にしたYES/OPSが作成されている．

また，プロダクション・システムで多くの時間を要する条件部の照合の操作（マッチング処理）を，OPS5ではRETEアルゴリズム[12][13]を用いて，高速化している．また，OPS83やYES/OPSではこのアルゴリズムの改良によりさらに高速化を行っている．本システムでは，このアルゴリズムを時間情報を含むルールの推論のために拡張した時間RETEアルゴリズムを用いて，時間条件のマッチング処理の高速化を行った．この拡張はネットワークの構造と，ネットワーク上の処理手続きの変更からなる．ただし，この拡張はあくまでも時間条件を持つ推論を考慮したものであり，一般のプロダクション・ルールの実行では，実行効率は従来のRETEアルゴリズムとほとんど変わらない．

また，ルールの記述性を高めるためオブジェクト指向[8]の考え方を取り入れた．まず，クラス，インスタンス間の継承による階層的なモデル構築を可能にした．また，オブジェクト間の通信リンクを定義し，リンクによるメッセージ通信によるオブジェクト間の関係の記述も可能とした．このような考え方で物理系のモデル化を行ったシステムとして，OPELLIA[18]があるが，OPELLIAでは離散時間を含む推論は考えられておらず，また，オブジェクト化の方法やリンクの記述法も異なる．

本章では２節でプロダクション・システムOPS5とRETEアルゴリズムについて述べ，３節では離散時間プロダクション・システムの仕様について述べる．そして，４節で，本章で提案する新しい時間RETEネットワークの構造について述べる．５節では，オブジェクト指向アプローチの導入について述べ，６節でOPS5およびRETEアルゴリズムとの比較を行う．

２．２　プロダクション・システム：OPS5[11]

２．２．１　プロダクション・システム

プロダクションシステム(PS)はワーキングメモリー(WM)，プロダクションメモリー(PM)，推論エンジンの３要素からなる．WMとは，推論状態を格納するための作業領域でワーキ

ングメモリーエレメント(WME)がそれぞれの要素を示す．PMはif-then-ruleの形式であたえられたルールベースである．PSの動作はWM，PMを参照し，条件部が特定のWMEとマッチングするルールの集合（コンフリクトセット(CS)）を作成し，その集合から１つのルールを選択し，そのルールの実行部を処理するというサイクルからなる．一般に実行によりWMが変更され，新たに得られたWMに対して同じサイクルを繰り返し実行して推論を進める．このようなサイクルを認知-行動サイクルと呼ぶ．

OPS5においては，WMEは次のような形式をとる

```
(class_name  ^attribute_1     value_1
                :               :
             ^attribute_n     value_n  )
```

具体例を示すと次のようになる．

```
(pipe  ^num 5  ^flow 7 ^status normal)
```

またルールは次のような形式をとる．

```
(P  tank-rule
(pipe ^num <X> ^flow <Y> ^status normal)
(tank ^num <X> ^qu <Z>)
->  (modify 2 ^qu compute(<Z>+<Y>) ) )
```

条件部は基本的にはWMEとおなじ形式で書かれるが，<X>のような変数を用いることができる．また実行部は，action名、条件節番号、およびattribute名と値の組のリストからなり，条件節番号で与えられた条件部にマッチングするWMEに対して作用する．この

例では，「 ^numが一致するpipeとtankに対して，pipeの^statusがnormalならば，tankの^quをpipeの^flowの分だけ増加させる．」というルールを示している．

２．２．２　RETE アルゴリズム[12]

PSの認知-行動サイクルにおいて，最も時間を要するプロセスは，前節に述べたマッチングを行い，CSを決定するプロセス（照合プロセス）である．RETE アルゴリズムとはこの処理を効率的に行うために，OPS5において用いられているアルゴリズムである．RETEアルゴリズムの基本的な考え方は，マッチングにおける無駄な繰り返しを避けることで，その効率を向上させるというものである．そのためには，一般的なPSの実行に見られる，次のような特徴を利用する．

(1)１サイクルの実行では，WMの一部のみが変化するという一時的冗長性(Temporal Redundancy)

(2)ルール群は類似するパターンを数多く，含んでいるという構造的類似性(Structual Similarity)

RETEアルゴリズムでは，これらの特徴を生かすために，ルールの条件部をRETEネットワークと呼ばれるデータフローネットワークにコンパイルする．更にネットワークの各ノードにマッチングに関する情報を格納することにより，マッチングの回数を最小限にしている．

tank-ruleに対するRETEネットワークをFig.2.1に示す．ネットワークの各ノードは照合条件に，アークはデータの流れを表現している．ノードは次の４種に分類される．

(1)Root

(2)1-入力ノード　各属性に関する条件のマッチング

(3)2-入力ノード（βノード）条件部の要素間の結合に関する照合，および変数の一貫性のチェック（βマッチ）

(4)　終端ノード

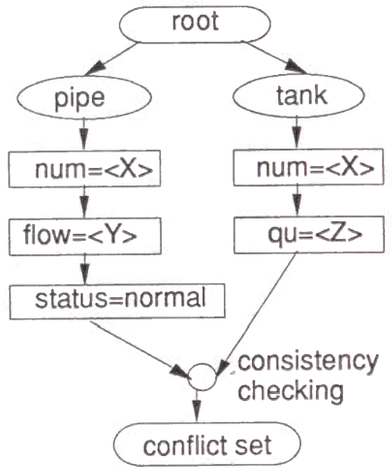

Fig.2.1 Rete network

WMの状態は，2-入力ノードの入力部において条件を満たすWMEを記憶することによりネットワーク上で保持される．

WMの更新は，ネットワークのrootノードからトークンを送り出すことにより行われる．トークンはWMに付加(make)および削除(remove)されるWMEに対応し，付加のときは記号'+'を削除のときは記号'-'を付けたWMEのリストにより表される．またWMEの変更(modify)の際は，元の内容を削除した後に新たにWMEを付加する．

例えばtank-ruleに対し(1)に示すWMEが付加されると，(2)のトークンがrootノードに入力される．そしてトークンはネットワーク上を流れ，その際に変数に対する束縛とその一貫性のチェックが行われ，最終的に(3)に示すようなトークンが終端ノードに出力される．


```
(1)  (pipe ^num 1 ^flow 5 ^status normal)
     (tank ^num 1 ^qu   0)
(2)  +(pipe ^num 1 ^flow 5 ^status normal)
     +(tank ^num 1 ^qu   0)
(3)  +< (pipe ^num 1 ^flow 5 ^status normal),
        (tank ^num 1 ^qu   0) >
```

このように、マッチングは各サイクルで変更されたWMEに対してのみ行われるため，一時的冗長性が有効に利用される．また，ルール群におなじ属性に関する条件部を持つものがあるとき，その条件部は一つの1-入力ノードにコンパイルされるため，構造的類似性も利用される．OPS5では，このようなRETEアルゴリズムにより高速なマッチング処理を実現している．

ただし，OPS5のルール文が時間を直接には表現できないために，そのままでは時間を含む事象の推論を行うことはできない．また，時間を制御するルールを加えて，それを用いて処理する方法も考えられるが，その場合は時間処理に伴うオーバーヘッドが大きくなり，効率的でない．そこで時間処理を効率的に行うために，４節で述べる時間RETEネットワークアルゴリズムを考案した．

２．３　離散時間プロダクション・システム

本章で扱う離散時間プロダクションシステムはOPS5と同様に，ルールベースで推論を行うものである．一般的なプロダクションシステムと異なるのは，ルールに離散時間情報を加えられるようにし，時間を含む事象を記述できるようにしたことである．OPS5では，認知・行動サイクルに基づくステップはWMにタイムタグとして付加されるが，このステップは単に１ルール実行されたという区切りであり，実時間とは対応しない．本システムでは，認知・行動サイクルに加え，離散時間ステップを実時間に対応するステッ

プとして独立に用いることで，各離散ステップ毎にそのステップにおけるワーキングメモリーの変更を全て行った後，次の離散ステップに移るという処理を行う．具体的には，各離散ステップ内で認知・行動サイクルを実行し，そのステップ内で実行可能なルールがなくなったとき，次のステップに移る．ここで，時間に関しては連続時間を扱うようなシステムも考えられるが，処理するデータが定性値という離散的なものであるため，時間のみを連続的にしても意味がないため，離散時間で時間情報を表すものとする．

このような離散時間ステップをルール内で記述するため，次のような機能を設けた．

① 各属性に与える条件に対して，条件生起時間（条件が成り立つまでに要する時間）と条件消滅時間（条件が満たされなくなる時間）を与えられる．
② 実行部に対して，条件部が成り立ってから実際に実行されるまでの時間（delay time)を与えられる．

①については，条件節の形式をBNFで次のように定義する．

```
<rule>::=
(p rule-name
 <condition-clause> ->
       <action-clause> )
<condition-clause> ::= (object-name <condition-list>)            ･･･(a)
<condition>::= ((<attribute-condition>)   <time-condition>)      ･･･(b)
             |  <attribute-condition>
<attribute condition> ::= attr-name  <conndition-type> value
<condition-type>::= empty | = | < | > | <> |                     ･･･(c)
```



```
<time condition> ::=  activate-time(lwt)  inactivate-time(upt)
    |  activate-time(lwt)
    |  0                    inactivate-time(upt)                ･･･(d)    (1)
```

conditionは，それぞれ一つの属性に対し，(b)の形式で属性値条件と時間条件の１組か，属性値条件のみが与えられる．属性値条件はOPS5と同じ形式で与え，時間条件も持つ場合にのみ () で組とする．(d)は属性値条件に対する時間条件を与え，属性値条件(c)が成り立ってからその条件節(b)が活性化されるまでの時間lwt(条件生起時間）と条件節が成り立たなくなるまでの時間upt(条件消滅時間）をステップ数で与える．つまり，条件節は属性値条件が満たされてlwtステップからuptステップの間だけ活性化され，全ての条件節が活性化されたルール文の実行部が実行される．また，時間条件を与えない場合は，lwt=0，upt=∞として処理され，時間条件を１つのみ与えた場合は，その値はlwtとして扱われ，upt=∞となる．条件としてuptのみ与えたい場合は，lwt=0として時間条件を記述する。

本システムの対象とする推論は離散時間をステップとして扱うものであり，このような条件記述を行うことで，ある事象が起こってから数ステップ後に他の事象が起こる場合や，ある事象の影響が数ステップ後に消える場合等を記述できる．また，Temporal-logicなどで行われている事象間の因果関係の記述は，(1)複数の事象に対する条件節を１つのルールの条件部にする，(2)因果関係を記述する変数を用いる，(3)オブジェクト間のメッセージ伝達を用いる，などの方法で可能なため，特に用意していない．

②については①と同様に実行部に対して遅れ時間（delay time）を与えることで実現する．実行部の形式をBNFで与えると次のようになる．

```
<action-clause> ::= (<action>  delay-time)                      ･･･(a)
                  | <action>
<action>  ::=(<action-type>  <action-parameter>)                ･･･(b)
```

```
<action type> ::= make | modify | remove | etc.                    ･･･(c)
<action-parameter> ::=<object> <parameter-list>                    ･･･(d)
<object> ::=| object name | clause number |                         ･･･(e)
<parmeter>::= attr-name  new-value                                 ･･･(f)     (2)
```

基本的にはOPS5と同じだが，(a)の形式でdelay-timeが与えらる．actionの作用先は(e)の形式で，条件節の番号，あるいはオブジェクト名を直接に指定して与え，条件節番号の場合は条件にマッチングしたオブジェクト，オブジェクト名の場合はその名前のオブジェクトが作用の対象となる．また，valueとしては定数，変数の式の他に属性値に対する変化量を与えられるようにした．これは，次の形式で元の値に対して行う作用を記述するのに用いる．

```
((modify  test  ^a   compute(#+1)) 1)
((modify  2  ^b compute(#+1) ^c compute(#-3)) 1)     (3)
```

'#'が元の値を表し，それに続く式が変化を示す．具体的には，「条件が成り立っている間，１ステップ毎にある属性の値が増加する．」等の事象を記述するのに用いる．

実行部は，そのルールの全ての条件部が満たされてから，delay-timeで与えられたステップ後（与えられていないときはすぐに）初めて実行される．その後，条件部が満たされている間，１ステップに１回ずつ実行される．ただし，１回目の実行後，新たにWMEに対し変化を起こさないactionは，その後のステップでは実行されない．また，条件部が満たされなくなると，新たに条件部が満たされるまで，実行されない．

このようなシステムを用いて，バルブ付バッファタンクのモデルについて記述した例を次に示す．


```
/* object definition */
/* (tank ^LE  ^Fin  ^Fout ^V1 ^St )  */
/* 実際は後に述べるオブジェクト形式で定義される     */
/* end of definition */

(p set-level
  (tank ^Fin <X> ^Fout <Y> ^St normal)
   -->    ((modify 1 ^LE compute(#+<X>-<Y>)) 1))                    ①

(p valve-open
  (tank ((^LE > 110) 2))
  -->     ((modify 1  ^V1 2) 1))                                    ②

(p valve-close
  (tank ((^LE < 90) 2))
  -->     ((modify 1  ^V1 0) 1))                                    ③

(p valve-normal
  (tank ((^LE { > 89 < 111} ) 2))
  -->     ((modify 1  ^V1 1) 1))                                    ④

(p set-Fout
  (tank  ^LE <X>  ^V1 <Y>)
  -->     ((modify 1 ^Fout compute(<X>*<Y>/10)) 1))                 ⑤
```

クラスtankは５つの変数を持ち，LEがタンクのレベル，Finが流入量，Foutが流出量，

V1がバルブの開度、Stが故障などの状態を示す．このシステムはLE=100,Fin=10,Fout=10,V1=1のとき平衡状態になる．ルールの説明を行うと、①については，OPS5の例であげたルールと同じであり，各ステップで１回ずつ実行される．②③④はバルブのコントロールルールである．バルブは0,1,2の3つの状態を持ち0のときは閉じており，1のときは通常の開度，2のときはcその2倍の開度になる．また，コントロールルールはタンクのレベルが100付近に留まるように働く．②のルールはタンクのレベルが2ステップの間，110を越えるとバルブを開度2まで開けるもので，③はその逆にタンクのレベルが2ステップの間，90を下回ったときにバルブを閉じるものである．また，④のルールはタンクのレベルが2ステップの間，90から110の間にあったときに，バルブの開度を1に戻すものである．⑤のルールはタンクのレベルとバルブの開度から流出量を決定するルールである．このように，本システムでは離散ステップに従い実行されるモデルを，各変数の関係を記述することで，宣言的に定義することが可能である．

本章では，このようなシステムを離散時間プロダクション・システムとして用いる．

２．４　時間RETEアルゴリズム

前節で述べたような時間を含むプロダクションルールを，従来のRETEアルゴリズムを用いて実行すると，時間条件の処理が効率がよく行われない．（詳細は５節）そこでRETEアルゴリズムを改良し，離散時間推論を行うネットワークアルゴリズムを開発した．今後このアルゴリズムを時間RETEアルゴリズムと呼ぶ．

このアルゴリズムは，RETEアルゴリズムと同様にマッチングの効率をあげるため一時的冗長性，構造的類似性を利用し，さらに離散時間の処理をネットワーク上で行うものである．

２．４．１　トークンの構成

はじめに，時間RETEアルゴリズムで用いるトークンについて述べる．actionの実行に伴うWMEの変更は，RETEアルゴリズムと同じくトークンとしてネットワーク上に伝達されるが，このアルゴリズムでは次の２種類のトークンを用いる．

（１）　action トークン

actionトークンは次のような要素からなる．

a) class name:　actionが作用するclass名

b) action type: 実行するactionの型．ただしWMEに対して作用するactionのみがトークに変換される．具体的には次の３つである．

1)　make:　　新たにWMEを付加する．

2)　modify:　すでに存在するWMEを変更する．

3)　remove:　あるWMEを削除する．

c) parameter: actionに対するパラメータ．

（２）　elementトークン

elementトークンはRETEアルゴリズムであつかうトークンに相当し，actionにより作用されるWMEとactionの型を示す記号からなる．

```
((make class1  ^a 1  ^b 1))
     ···  +(class1 ^a 1  ^b 1)
((remove class1  ^a 1  ^b 1))
     ···  -(class1 ^a 1  ^b 1)
((modify 1 ^a compute(#+1)  ^b 1))
   ただし，(class1 ^a 1  ^b 0)が条件節１にマッチ
     ··· #(class1 ^a 2  ^b 1)
```

ただし，RETEネットワークでは記号としては，付加を表す"+"と削除を表す"-"の２つのみであったが，このアルゴリズムでは変更を表す"#"を加え３つの記号を用いる．これは，RETEアルゴリズムではmodifyをmakeとremoveの組合わせで行っていたのに対し，このアルゴリズムではmodifyを独立な処理で行うためである．

２．４．２　時間RETEネットワークの構造

RETEアルゴリズムでは条件部をRETEネットワークにコンパイルしてマッチング処理を行っている．このアルゴリズムでも同様に，時間条件を含む条件部のコンパイルを行い，ネットワークに変換する．このネットワークを時間RETEネットワークと呼ぶ．次に時間RETEネットワーク(以後，本章ではネットワークと記す）の構造について述べる．ネットワークの概要をFig.2.2に示す．

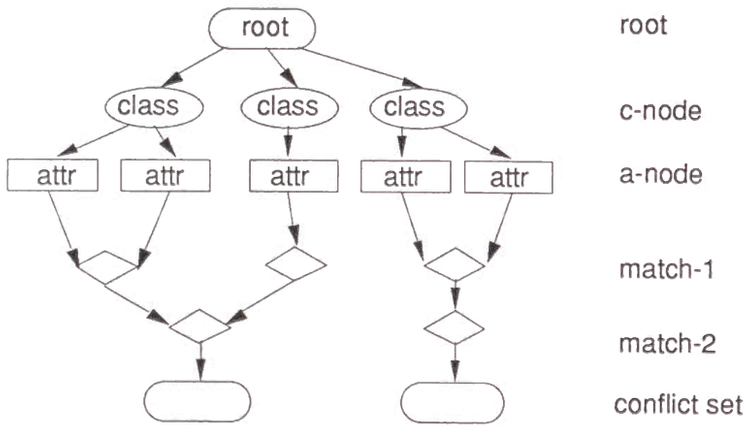

Fig.2.2 Structure of the network

このネットワークは，５種類のノードをもち，各々データフローアークにより，結合される．各ノードではその種類に応じて条件のマッチングが行われ，条件を満たすデータのみ次のノードに送られる．各ノードでの処理は次のようになる．

root:　競合集合より作成されたactionをc-nodeへ送る．

c-node:クラス名のマッチング，およびactionの実行．

a-node:各属性に与えられた条件の中の１つのマッチング,および条件部に与えられた時間条件の処理．

match-1:属性値をWMEごとにまとめ，変数の一貫性をチェックする．

match-2:１つのルール条件部をまとめ，変数の一貫性をチェックし，競合集合を更新する．

RETEネットワークとの構造的な違いは，RETEネットワークでは１つのクラスに対して属性に対する条件を直列に展開していたのに対し，このネットワークでは並列に展開することによって，各属性ごとにa-nodeとmatch-1の間のアーク上で時間条件を与えられる様にした点である．ここで時間条件とは，先に述べた条件生起時間lwt(lower limit time)，条件消滅時間をupt(upper limit time)の２つである．また並列展開のため分解されたWMEを１つのtokenにまとめるmatch-1が加えられた．

actionトークンはrootノードに入力され，actionに与えられた実行時間になると，クラス名に応じてc-nodeに送られる．そして，そこで実行され，elementトークンに変換される．また，a-nodeからmatch-1へのアークに対して各属性ごとの時間条件を与え，トークンの転送時に時間条件だけ遅らせて転送することで，時間条件のマッチング操作をなくしている．RETEネットワークのように直列に展開すると，トークンが各属性に対する条件ノードを順に通過するため，各属性に独立に時間条件を与えることが不可能である．また，ノードはそれぞれメモリーを持ち，マッチングの中間結果等が格納される．

さらに，実行時には属性値条件ごとにa-nodeが分かれているため，変更が生じた属性に関係するa-nodeにのみtokenを送るものとする．

次のルールに対するネットワークはFig.2.3になる．

```
(p test
 (t1 ((^a < 7) 2 6)  ^b 0)                                    (1)
 (t2  ^c 9)                                                   (2)
  --> ((modify 1 ^a compute(#+1)) 1)
      ((modify 2 ^c compute(#+1)) 1))        例1
```

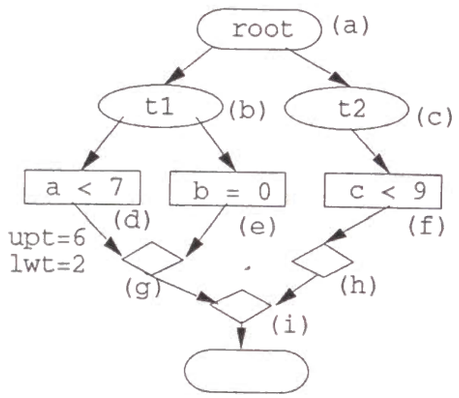

Fig.2.3 Example of the network

クラスt1(b)とクラスt2(c)が、c-nodeとして生成され，t1についてはルールの条件部(1)から(d)と(e)の２つのa-nodeがつくられ(d)と条件部(1)に対応するmatch-1(g)の間のアークに時間条件としてlwt=2,upt=6が与えられる．条件部(2)からはa-node(f)とmatch-1(h)がつくられ，条件部全体がmatch-2(i)でまとめられる．

また，RETEアルゴリズムでは１種類であったトークンが，このアルゴリズムではactionトークンとelementトークンの２種類を用いている．RETEアルゴリズムではrootノードにactionの実行の結果として変更されたトークンが入力される．しかし，本アルゴリズムが扱うdelay timeを持つactionの場合，トークンの生成時にはaction実行時のWMEの属性値はまだ不定なので，actionトークンとしてactionと実行先のWMEを指定し，c-nodeで実行時のWMEに対し処理を行う．そして，変更されたWMEからelementトークンを生成する．

ネットワーク上にこのようなトークンを流すことにより，各ノードにおいてメモリーの更新が行われ、さらに次のノードにトークンが送られる．この操作はノードにトークンが到着した時のみに行われる．また，elementトークンはWMEと一対一に対応し，実際にはelementトークンからのポインターによって結合されている．

２．４．３　マッチング処理

RETEアルゴリズムのマッチング処理では，１入力ノードにおいて'+'モードのトークンに対して，属性値のマッチングを行い，その結果に従いノードのメモリーを変更する．本アルゴリズムでは，modifyモードのトークンの処理および時間条件の処理を行うためマッチングの処理を一部変更した．

(1)modifyモードのトークンの処理

a-nodeにおける属性値のマッチング処理において，　'#'(modify)モードのトークンに対しては次のような処理を行う．

１.属性値に対する条件のマッチングを行う．条件を満たす場合２へ，満たさない場合５へ

２.a-nodeのメモリーに同じWMEを表すトークンが存在するか調べる．存在すれば３へ，存在しなければ４へ

３.条件部が変数を含む時は，そのトークンの変数束縛情報を変更する．そうでないときはそのまま終了．

４.'+'モードのトークンとおなじ処理を行う．つまりトークンをメモリーに登録し，変数束縛情報を作成する．

５.a-nodeのメモリーに同じWMEを表すトークンが存在すれば，メモリーから取り除き，modeを'-'に変更してmatch-1へ送る．

後に述べる時間条件の処理で，トークンのa-nodeへの滞在時間を用いるため，modifyによる値変更の際に条件が継続して満たされている場合は，２，３のように処理することでトークン自体の書換えは行わないようにしている．modifyをremoveとmakeの組合わせで行うと，トークンの削除により滞在時間の情報が失われるために問題が生じる．また，５の処理により条件を満たさなくなったWMEを含む要素を，競合集合から取り除く．このようなmodify命令の処理の改良は，属性条件のマッチングに関する高速化にも寄与する．また，modify命令に関しては，YES/OPSも同様の処理で高速化が行われているが，次に述べるような時間条件の処理についてはYES/OPSは時間条件を扱うシステムではないために考えられていない．

(2)時間条件の処理

先に述べたように、a-nodeとmatch-1間のアークに２つの時間条件(lwt,upt)が与えられる．トークンはメモリーに付加されてからlwtステップ後（lwt=0の時は即座に），その出力アークを通してmatch-1へ送られる．そして，uptが条件部に与えられている場合は，uptステップ後にそのトークンがメモリーに存在するならば，モードを'-'に変更してmatch-1へ送る．この操作により時間条件を満たさなくなったWMEを競合集合から取り除く．

ただし，各ステップにおけるトークン移動は，実際にはスケジューラにより行われ，各ノードでは，現在の実行ステップに関する情報を必要としない．つまり，a-nodeへトークンが到着したときに，そのトークンに対する時間条件をスケジューラに登録し，時間条件が満たされた時点でスケジューラからの指示により，a-nodeにおけるトークンの送出や削除が行われる．

ただし，属性値条件が変数を含む時は，a-nodeのみでは条件が確定しないため，トークンの送出を行うステップを決定できない場合がある．そのため，条件が変数を含むa-nodeでは，トークンがc-nodeから到着すると，時間条件lwtに関係なく，変数束縛情報

をmatch-1へ送る．そして，match-1,node5において変数の一貫性が満たされると，a-nodeに条件が満たされたことを伝え，前に述べた時間条件の処理が行われる．

WMEの変更に伴い，トークンに対する変更が行われても，a-nodeの属性値条件が継続して満たされている場合，競合集合の変化は起きないので，新たに変更されたトークンをmatch-1へ送る処理は行われない．ただし，その属性値条件が変数を含むときに限って，新しく生成された変数束縛情報をmatch-1へ送る．この操作により，変数値の変更に伴う条件間での変数の一貫性が調べられる．

２．４．４　アルゴリズムの実行例

この節では，例１のルールについて実行例をあげて解説する．初期条件を与えるために次のactionを実行する．

```
((make t1  ^a 4  ^b 0) 1)
((make t2  ^c 2 ) 1)
```

次に各ステップにおける実行を追っていく．

```
initialize  ((make t1  ^a 4  ^b 0)  1)        (A)
            ((make t2  ^c 2 )  1)             (B)          (1)
step1       +(t1  ^a 4   ^b 0)                (A)
            +(t2  ^c 2 )                      (B)          (2)
step2
step3       <(t1  ^a 4   ^b 0),(t2  ^c 2)>                 (3)
            (modify t1  ^a compute(#+1))      (A)
            (modify t2  ^c compute(#+1))      (B)          (4)
```

```
step4       #(t1  ^a 5   ^b 0)          (A)
            #(t2  ^c 3)                 (B)          (5)
step5       #(t1  ^a 6   ^b 0)          (A)
            #(t2  ^c 4)                 (B)          (6)
step6       #(t1  ^a 7   ^b 0)          (A)
            #(t2  ^c 5)                 (B)          (7)
            -(t1  ^a 7   ^b 0)                       (8)
```

これらのactionはどちらもdelay timeを持っているため，初期設定の段階ではそのactionに相当するactionトークンが(1)のようにrootノードに生成されるだけである．第１ステップでは(1)-(A)のactionトークンをノード(a)から(b)へ送る．(b)において，actionトークンは実行され，その結果生成されるelementトークン（(2)-(A)）は(d)と(e)に送られ，属性値条件のマッチングが行われる．(d)では属性値マッチングが成功するが，アークに対し条件生起時間(lwt=2)が与えられているため，２ステップ後に(d)から送り出される．(e)でもマッチングは成功し，トークンは(g)に送られる．(g)ではクラスに対する全条件部が満たされるまでトークンは処理されないので，(d)からのトークンの到着を待って処理される．次に(1)-(B)のactionトークンを(c)に送り，そこで実行し，生成されたelementトークン（(2)-(B)）は(h)，(i)と送られる．(i)では(g)からのトークンの入力があるまで，処理が待たされる．

第２ステップでは変化がなく，第３ステップで時間条件が満たされ，(d)から(g)にトークンが送られる．ここで，(g)の全条件が満たされトークンは(i)へ送られ，同様に全条件が満たされ２つのelementトークンの組(3)がこのルールに対する競合集合の１要素として与えられる．さらに，その要素に対して実行部からactionトークン(4)が生成され，rootノードに送られる．

第４ステップでは(4)のactionトークンが処理される．(4)-(A)は(b)で実行され，elementトークン(2)-(A)が(5)-(A)に変更され，(d)においてマッチングされる．マッチングは成功するがトークンのモードがmodify('#')で，かつ条件部に変数を含まないので，

(g)にトークンを送る操作は行わない．(4)-(B)についても同様に処理され，elementトークン(2)-(B)が(5)-(B)に変更され，処理される．

第５ステップでは第４ステップと同じくactionトークンが実行され，elementトークン(6)が得られる．第６ステップでelementトークン(7)が得られる．このとき，elementトークン(7)-(A)が(d)におけるマッチングで失敗し，トークンのモードをdelete(-)に変更し(g)へ送る．(g)ではそのトークンをメモリーから取り除き，さらに(i)へ送る．(i)で入力バッファからトークンを，さらに第２ステップで生成されたそのトークンを含む競合集合要素を取り除く．競合集合要素の削除と同時に，その要素から生成されたactionトークン(4)が削除される．この時点で実行可能なactionトークンはなくなり，実行は終了する．

次に，おなじネットワークに対して次ののactionを用いて，初期条件を与えた場合について考える．

```
((make t1  ^a 2   ^b 0) 1)
((make t2  ^c 2)  1)
```

第１ステップから第５ステップまでは前の例と同様に実行され，elementトークン(1)が得られる．

```
step5       #(t1  ^a 4   ^b 0)          (A)
            #(t2  ^c 4)                 (B)          (1)
step6       #(t1  ^a 5   ^b 0)          (A)
            #(t2  ^c 5   ^b 0)          (B)          (2)
step7       -(t1  ^a 5   ^b 0)                       (3)
```

第６ステップでは，マッチングが成功しelementトークン(2)が得られる．第７ステップ

でもマッチングは成功するが，elementトークン(3)-(A)が(d)と(g)の間に与えられた条件消滅時間(upt=6)により，時間条件を満たさなくなり，(d)からモードをdelete('-')に変えて(g)に送り，前の例と同じく競合集合を更新する．

２．５　オブジェクトの実現

２．５．１　記述言語のオブジェクト化

プラントのような大規模なシステムでは，数多くの要素が結合されて構成されている．その中にはまったく同じ要素が複数あったり，部分的に同じ要素がいくつか存在する．そのようなシステムのモデルを作成するには，オブジェクト指向アプローチがよく用いられる．つまり，各部品をオブジェクトとして定義し，その結合によりシステム全体を表現する．また，各オブジェクトはクラス，インスタンス間の継承関係を用いて階層的に定義される．本研究では、そのような知識記述を離散時間プロダクションシステムに組み込む方法について考察を行った．具体的には，次のような改良を行った．

（１）クラス，インスタンス間の継承関係による，階層的な知識表現が可能である．
（２）オブジェクト間のリンクとその通信機構により，オブジェクト間の関係を表現することが可能である．

まず，（１）に関しては次のような表現でオブジェクトの定義を行う．


```
class:
  or            object name;                          (a)
instance:
super:          list of super classes;                (b)
attributes:     list of instance-attributes;          (c)
links:          list of link-definition               (d)
rules:          list of instance-rules                (e)
end; / end.
```

(a)はオブジェクト名の宣言で，classまたはinstanceのいずれかとして宣言する．ここでclassとは概念の集合であり，instanceが実際の物に対応する．(b)はそのオブジェクトの上位クラスの宣言で，複数クラスを親とする多重継承も可能である．(c)はそのクラスに属するインスタンスの持つ属性の宣言である．これらの属性はclassでは実体を持たず，その下位の全てのinstanceが属性を継承し，その値は各instanceが別々に持つものである．(d)はリンクの宣言で，後に詳細を述べる．(e)はそのクラスに属するインスタンスの持つルールの宣言である．ルールは属性と同じく，classにおいては作用せず，その下位の全てのinstanceに継承され，そこで有効になる．また，クラスにおけるルールの宣言時には，自分自身を示すインスタンスのオブジェクト名は'*'を用いて表す．

（２）については，次の形式でオブジェクト間のリンクを記述を行う．

```
links: list of link types                             (f)
links: list of link definitions                       (g)
<link-definition> ::= <link-type> -> <instance-name>
```

(f)はclassにおけるリンクの記述で、その下位のinstanceで用いるリンク型の中でそのクラスに関係するものを定義する．(g)はinstanceにおけるリンクの記述で，リンクの型とその結合先のinstance名を定義する．リンクはオブジェクト間のメッセージ通信の際に相手のオブジェクトを直接指定するのではなく間接的に指定するのに用いる．この方法により，上位クラスにおけるルール記述ではメッセージの送り先をリンク型を用いて抽象的に表現し，実行時に，オブジェクト間の関係に基づいて設定されたリンクを用いてメッセージの受け手となるオブジェクトが決定される．

実際にメッセージ送信を行うには，send命令を用いる．次にその構文を示す．

<message-sending> ::= (send <message-clause> delay-time) (h)

<message-clause> ::= <destination> <message-pattern> (i)

<destination> ::= <object-name> | <link-type>':' (j)

<message-pattern> ::= <action-clause> | <parameter-message> (k)

<parameter-message> ::= ((selector <message-selector>) <parameter-list)) (l)

<parameter-list> ::= {(<parameter-name> <value>)} (m)

send命令には(h)(i)の形式で，その送り先とメッセージパターンを与える．送り先は(j)のようにオブジェクトを直接に指定する方法と，リンクを用いた間接指定で行う方法がある．リンクを用いた場合は，send命令が実行されたオブジェクトから，link-typeで指定されたリンクで結合された全てのオブジェクトにメッセージが送られる．メッセージパターンには(k)のようにactionで与えるものと，(l)の形式のメッセージセレクタとパラメータリストの組で与えるparameter-messageがある．actionを与えた時は，その送り先で与えられたactionが実行される．この場合は送り先のオブジェクトの情報が必要となるため，送り先のオブジェクトと送信を行うオブジェクトは同じクラスを親とし，actionに伴う作用はその親クラスで定義された変数を対象とする必要がある．paramete

r-messageは，一般的なオブジェクト指向システムにおけるメッセージによるメソッド起動にあたるものである．ただし，本システムではメソッド定義の方法は特別には用意されておらず，次のようにルールの条件部でメッセージの解釈が行われる．

<message-condition-clause> ::=

(link-type: ((selector <selector-name>) <parameter-matching-list>)) (n)

<parameter-matching-list> ::= { (<parameter-name> <variable> | <const>)} (o)

(n)の部分ではリンクタイプ，メッセージセレクタのマッチングを行う．ただし，全てのリンクタイプに関して有効なメッセージの場合はリンクタイプは'*'で指定する．また，(o)ではパラメータ部のマッチングを行い，ルール内変数に代入も行われる．このようにメッセージのマッチングが成功したルールの実行部が実行されることで，メソッドの起動の役割を果たしている．

オブジェクトによるモデル記述の例を次に示す．この例では，物流系としてのFlowObjectを親クラスとして定義し，その子クラスとしてPipe,Valveを定義している．また，フローを表すリンクタイプとしてF1を用意し，リンクによるメッセージ通信でオブジェクト間の作用を表している．

```
class:FlowObject;
attributes: Flow:str;
links:F1;
rules:
(p FlowRule
  (*  ^Flow <X>)
    ->(send *:F1 ((modify * ^Flow <X>) 1))
)
```

```
end;

class:Pipe;
super:FlowObject;
end;

class: Valve;
super:FlowObject;
attributes: Stat:str;
rules:
(p ValveClose
 (* ^Stat close)
    ->((modify *  ^Flow F-) 1)
)
(p ValveOpen
 (*  ^Stat open)
    ->((modify *  ^Flow Normal) 1)
)
end;

class: Pump;
super:FlowObject;
attributes: Stat:str;
rules:
(p PumpOff
  (*  ^Stat Off)
      -> ((modify *  ^Flow F-) 1)
)
(p PumpOn
  (*  ^Stat On)
      -> ((modify *  ^Flow Normal) 1)
)
end;
```


２．５．２　オブジェクトの組み込み

４節で述べたように，このシステムでは改良されたRETEアルゴリズムを用いることで時間条件のマッチングを高速化している．しかし，この節で述べているオブジェクトに関する処理をプロダクションルールで行うと，処理のオーバーヘッドが大きいと考えられる。そこで、オブジェクト形式で記述された知識をネットワークに組み込む方法について考察する．

まず，オブジェクト表現における上位クラスからの属性の継承のためにトークンの構成を変更する．ＯＰＳ５ではトークン上の各属性値に対し，番号づけを行い各ノードでの属性値の参照を高速化している．このシステムではオブジェクトに対応するトークンへの番号付けは次のような方法で行うものとする．

属性の番号付け

instance属性の番号付けは，最上位のsuper classで定義された属性から順に番号付を行う．ただし多重継承の場合は，上位クラスのリストに記述された順番に深さ優先で番号を付ける．

ただし，この方式でinstance属性に番号付を行うと，多重継承された属性では番号が各instance間で異なってくる可能性があり，マッチングノードにおける属性値参照の際になんらかの手段を構じなければならない．つぎに属性の番号が変わる場合の例を示す．

```
class : A;
attributes : a;
  :

class: B;
attributes : b;
  :
instance: C;
super:  B;              instance属性はbのみ，bの番号は１
  :
instance D:
super: A,B;             instance属性はa,b，bの番号は２
  :
```

次にルールの継承の方法を述べる．ルールの継承はFig.2.4のように上位クラスで定義されたルールを表すサブネットワークの属性のマッチングを行うノード(a-node)に，インスタンスからアークを結合することで行う．

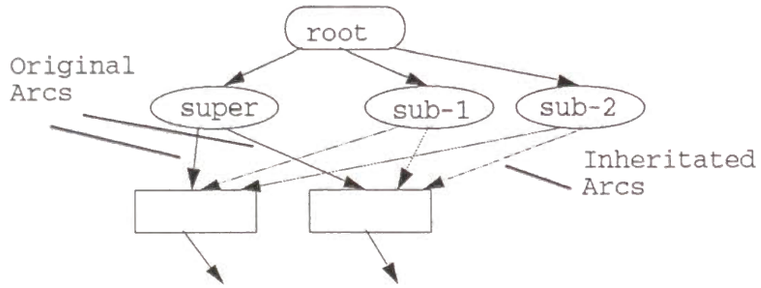

Fig.2.4 Inheritance of the rules

つまり，このアークを用いてインスタンスをあらわすトークンは上位クラスに送られ、ルールのマッチングが行われる．各クラスで定義されたルールには，そのクラスの下位にある全てのインスタンスからトークンが送られてくるが，match-1で各インスタンスごとにまとめられるため，他のインスタンスのマッチング結果と混同されることはない．ここで，上述のような属性の番号づけを行うことで，単純継承の場合はa-nodeでのマッチング時に上位クラスの属性の番号は等しくなる．ただし，多重継承の場合には，instance間での属性番号のずれを補正しなければならない．そのために次に述べるようなオフセット機構をネットに付け加える．super classで定義されたinstance属性の番号と実際のinstanceでの番号の差は，instanceのコンパイル時に計算可能なので，その値をオフセットとしてルールの条件部を表すa-nodeとinstanceに対応するc-nodeの間のアークに与える．そしてa-nodeでマッチングを行う際に，トークンが入力されたアークに与えられたオフセットの分だけ，属性番号を補正してトークンの属性値を参照する．オフセットの例をFig.2.5に示す．

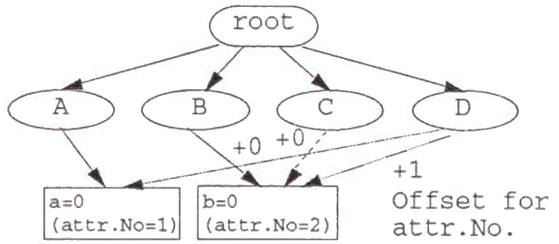

Fig.2.5 Example of the Offsets

また，オフセット値はトークンに付加して送られ，実行部で属性値を書き換える時も，同様にオフセットが使用される．

次に通信リンクとメッセージ通信機能のネットワークへの組み込みについて述べる．通信リンクは各オブジェクトに対応するc-node間のリンクとして生成される．メッセージの送り先がリンクの場合はこのリンクを通して間接的に，オブジェクトの場合にはそのオブジェクトに直接にメッセージが送られる．メッセージの送信にはsend命令を用い

るが，ネットワーク上ではc-nodeにおいてsend命令に対するactionトークンを実行する際に処理される．具体的には，メッセージがactionで与えられた場合は，そのactionを送信先にあたるc-nodeに送り，そこでそのactionが実行される．メッセージが，parameter-messageの場合には，次に述べるmessageトークンが生成され，送信先に送られ，そこでマッチングが行われる．ここでmessageトークンとは，parameter-messageをマッチングのために変換したもので，次のような要素からなる．

messageトークン

$(object-name <parameter-message>)

ただしobject-nameはメッセージを受け取るクラス名

ただし，messageトークンのマッチングは，elementトークンと同じく，ネットワークで行われる．

以上のような手法で，オブジェクトの導入に伴うプロダクションシステムの拡張部をネットワークに組み込むことによって，拡張に伴う処理のオーバーヘッドを最小限に押えることができると考えられる．

2．6　OPS5：RETEアルゴリズムとの比較

この節では，時間推論をOPS5で行った場合との比較について，具体例を挙げて述べる．4節で扱った例に対する，OPS5によるプログラムを次に示す．


```
(P time1-set
    (t1 ^a < 7)
    (time1 ^time -1)
    (systime ^time <X>)
    -->
      (modify 2 ^time <X>)
)                                          (1)
(P time1-reset1
    (t1 ^a > 6)
    (time1 ^time <> -1)
    -->
     (modify 2 ^time -2)
)                                          (2)
(P time1-reset2
    (systime ^time <X>)
    (time1 ^time < compute(<X>-5) )
    -->
      (modify 2 ^time -1)
)                                          (3)
(P inc-time
    (systime  ^time <X>)
    -->
      (modify 1 ^time compute(<X>+1) )
)                                          (4)
```

```
(P rule1
    (systime  ^time <X>)
    (time1  ^time { <= compute(<X>-2)
                    >= compute(<X>-5) } )
    (t1  ^b 0  ^a <Y> )
    (t2  ^c { <= 9  = <Z>} )
    -->
      (modify 3  ^a compute(<Y>+1))
      (modify 4  ^c compute(<Z>+1))
)                                          (5)
```

また，このプログラムから得られるRETEネットワークをFig.2.6に示す．プログラムでは(1)-(4)がクラス't1'の属性'^a'に与えた，時間条件を処理する部分で，クラス'time'の属性'time'に属性値条件が満たされた時間（離散ステップ番号）を記憶させ，(5)ではその値を使ってマッチングを行っている．(1)は属性条件を満たしたステップ番号を記録するルール，(2)は属性条件を満たさなくなったときのリセット，(3)は条件消滅時間の処理，(4)は離散ステップの更新であり，(5)が時間処理を行うルール本体となっている．OPS5には，時間処理を行う記述法が用意されていないため，(1)(2)(3)(4)のようにルールとして時間処理の準備を行う部分を加える必要があり，ルールの記述が理解しにくくなる．また，これらのルールの他にも，時間の処理と属性条件の処理のタスクを切り換えるルールも必要になる．

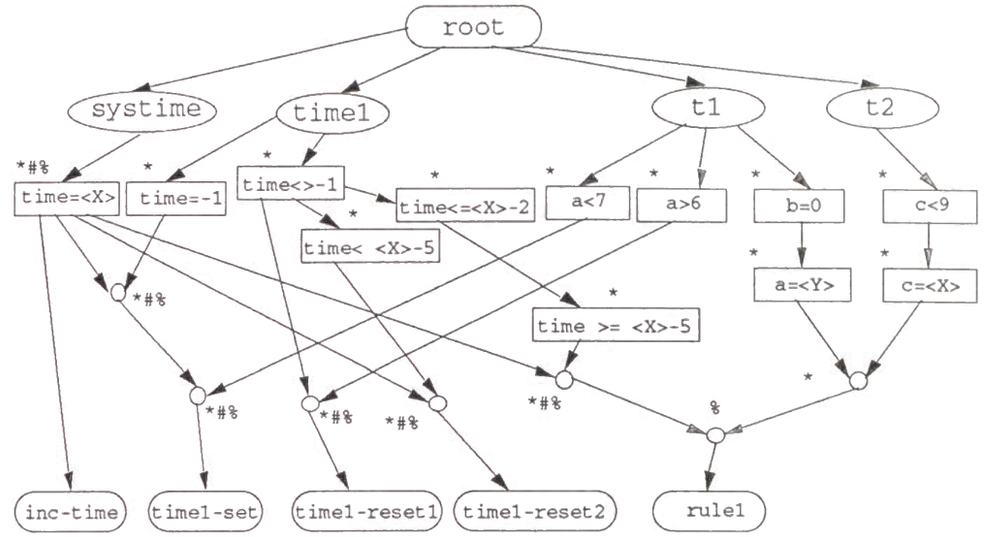

Fig.2.6 Example of Rete network

ネットワークの形状から見ると，OPS5のプログラムは，(1),(2),(3),(4)で示されるような時間条件の制御部が入るため、RETEネットワークはFig.2.6のように非常に複雑でノード数が多くなるが，４節で扱ったネットワークは単純でノードの数も少ない．

実行時のマッチング回数について比較すると，Table2.1のようになる．ただし（）の中の数値は実際に変数のマッチングが行われる回数（変数を持たない一貫性チェックを除いた回数）を示している．

	図17のRETEネットワークでのマッチング回数	図12のネットワークでのマッチング回数
属性条件が満たされるステップ (*)	属性マッチ 13 βマッチ 6 (2)	属性マッチ 3 Consistency 3 (0)
時間条件が満たされるまでのステップ (#)	属性マッチ 1 βマッチ 3 (2)	属性マッチ 0 Consistency 0 (0)
時間条件が満たされた後のステップ (%)	属性マッチ 7 βマッチ 3 (2)	属性マッチ 2 Consistency 0 (0)

Table 2.1 Number of matchings for examples

また，Fig.2.6の記号はTable2.1の中の各状況においてマッチングが行われるノードを示している．Fig.2.6で示されるように，クラスsystimeの関係するノードでマッチングや一貫性のチェックが多く行われており，処理時間の多くを費やすと考えられる．

また，時間条件１つ当たりのマッチング回数を比較したのが，Table2.2である．

	RETEネットワークでのマッチング回数	時間RETEネットワークでのマッチング回数
属性条件が満たされるステップ	属性マッチ 8 βマッチ 4 (2)	属性マッチ 1 Consistency 1 (0)
時間条件が満たされるまでのステップ	属性マッチ 1 βマッチ 3 (2)	属性マッチ 0 Consistency 0 (0)
時間条件が満たされた後のステップ	属性マッチ 3 βマッチ 3 (2)	属性マッチ 1 Consistency 0 (0)

Table 2.2 Number of matching for one time condition

OPS5では，属性条件が満たされるステップで時間条件の初期化を行うため，多くの属性マッチングが行われる．また，属性条件が満たされてから，各ステップで２つずつ，βマッチで変数のマッチングが行われている．これは時間条件のマッチングの際に，離散ステップ番号を変数に格納して時間条件と比較しているためである．時間RETEアルゴリズムではこの操作が行われず、システムに組み込まれているため，処理時間の短縮化が可能である．また，オリジナルのRETEアルゴリズムではmodify命令の処理が，removeとmakeの組合わせで行われるので，処理のオーバーヘッドが存在する．本アルゴリズムではmodify命令は独立に処理し，時間情報の更新等を行っているため，そのようなオーバーヘッドは生じない．

また，この例にあるような各ステップで一定量の変化を起こすような属性値に関して

は，modify命令で変化量を与える形式（３節(3)）を用いることで，効率の改善が行われる．このような場合，OPS5では各ステップで変数に属性値を代入し，それをcompute命令で計算し，属性値を変化させるという形式で記述される．そのため，各ステップでマッチング，競合集合の更新，modify命令の実行が行われ，その属性に関連するRETEネットワークのノード全てが動作する．本アルゴリズムでは，modify命令に対応するactionトークンが１ステップ毎に属性値を更新し，属性値条件が満たされている限りa-nodeより先のノードは動作させない．そのため，不必要な一貫性チェックや競合集合の更新は行われない．

２．７　結言

定性シミュレーションに用いる離散時間プロダクション・システムを定義し，その実行アルゴリズム，およびその実現法について考察した．今回作成したシステムは，次のような特徴を持つ．

１）離散時間を含む形でルールを記述できるプロダクション・システムを用いて，離散時間シミュレーションを行うことが可能である．

２）RETEネットワークを基にした，時間RETEネットワークにコンパイルして実行されるため，高速な実行が可能である．

３）オブジェクト形式のルール記述により，階層的な知識表現とリンクを用いたメッセージ通信による関係の記述を可能にし，柔軟なモデル作成を行うことができる．

しかし，このシステムによりプラントなどの物理系の解析に用いようとした時に，次のようなことが指摘された．

１）オブジェクトに対してマクロに状態を定義し，ルールおよびメッセージによりオブジェクト間の相互作用を記述するという形式のモデル化には問題はないが，オブジェクト内部をミクロに変数レベルで記述する場合には，変数間の作用を全てプロダクションルールで記述することは難しく，オブジェクト内の変数間の物理的な関係のみを記述することでモデル作成が可能なシステムのほうが使いやすい．

２）同じく変数レベルの記述においては，メッセージ通信によるオブジェクト間の相互作用の記述は難しく，リンクによるオブジェクトの結合のみで相互作用が自動的に実現される方がよい．

３）多重継承の際に，クラスの性質を継承して新たなクラスを作成することは可能だが，クラスを物理的に組み合わせて新たなクラスを作成することが必要である場合がある．

これらのことから，物理系の定性モデル記述とその解析のために，次章で述べる定性ネットワークによるモデル記述と４章の定性ネットワーク上の定性シミュレーションアルゴリズムについて考察を行った．

第3章　オブジェクト指向アプローチに基づく定性ネットワークモデル

3．1　緒言

前章では，プロダクションルールによる時間システムの記述とそのシミュレーションについて述べた．そのルールベースによるモデル記述は，汎用性は高いものであるが，物理系の記述に対象を絞ると，いくつかの改良すべき点が見られた．そこで，ベースモデルとしては定性ネットワークを用い，前章では従来の枠組みのままで用いていたオブジェクト指向アプローチに拡張を加えた．本章，および４章では物理系に対象を限定し，定性的にシミュレーションを行うシステムについて述べる．　定性シミュレーションに関する研究においては，そのモデルの記述法は，知識の記述性，およびシミュレーションの実行法に重要な関わりを持つ．これまで行われてきた研究では，主に定性方程式が記述モデルとして用いられてきた．[20] この方法では，実際の数値を扱う方程式を変換して定性方程式を作成するため，シミュレーションの対象が方程式として記述されている必要がある．しかし，プラント等の大規模な系では，対象の方程式を得ることは難かしい場合がある．また，方程式自身が，プラントの構造と直接に結び付いているものではなく，プラントのある部分を物理系として表すものであるため，プラントの構造に関する知識から直接に，定性モデルを得ることは困難である．前章で扱ったプロダクションルールによる記述は，プラントの各要素に対してマクロに状態を定義し，各状態間の関係をルールやメッセージで記述するというモデル化であるが，この方法でも構造的なモデルからの変換が必要になる．そこで，本研究では，プラントの構造から容易に導くことができる定性ネットワークをモデルとして用い，その構成とネットワーク上での定性シミュレーションについて考察した．本章では，そのネットワーク構築の枠組について述べる．

定性的なモデルを作成する際に，プラント全体に対して１度に組み上げていくのは困

難であり，何らかの系統的な手法が必要とされる．そのため，本研究ではオブジェクト指向アプローチに基づく継承の手法[18] により，段階的にモデルを作成する枠組みについて考察する．このようなモデル記述においては，オブジェクト指向システムの持つ要素の１つである継承の考え方が重要である．継承とは，上位の概念を表すオブジェクトの性質が，その下位のオブジェクトに受け継がれるという機構である．本研究では，プラントなどの構造的なオブジェクトをモデル化する際の枠組について考察した．これは，全体を部品から組み上げて行くために必要なオブジェクトの結合と，従来用いられてきた上位クラスからの性質の継承に加え，各部品をオブジェクトの結合として構成できる継承(コンポーネント継承）の実現の２点からなる．また，コンポーネント継承の導入はは多重継承[21] の拡張という点においても意味を持つものである．

定性モデル記述とそのシミュレーションに関してはQP理論およびSTEAMERで用いられている物理現象に基づきモデル化を行う方法[6] や，物理要素に基づきモデル化を行うEVISION[5] が有名である．ここで，プラントのような大規模物理系をモデル化するには，プラントの構造から物理現象に変換する必要がないため，物理要素に基づくモデルの方が作成し易いと考えられる．そのため，本章で述べるモデル記述法は，物理要素に基づいてモデル化を行う方法を用いるものであるが，改良されたオブジェクト指向アプローチを用いモデルの記述性を高めたものである．また，大規模プラントの定性的な取扱法についてB.Falkenhaier,K.D.Forbus[1988][22] の論文において，複数の視点に基づくモデル構築とそのシミュレーションについて述べられているが，大規模プラントに対しては整合性を持った視点を与えるのは困難であると考えられる．また，視点に基づくモデル化と本章の手法を融合させることで，モデルの入力を支援することも可能である．GensymのG2[23] においても階層的なモデルの記述が可能であるが，継承に関しては性質の継承のみであり，ルールまで含めたオブジェクト単位での構造化までは行われていない．また，意味ネットワークを用いて，has-a関係，is-a関係，part-of関係などを用いて記述する方法も考えられ，この方法でも結果的には同じ様な知識構造が得られるが，本研究は，各オブジェクトの結合を含む系統的な記述を目的とし，ソフトウェア構築の効率

化の立場から記述言語を作成するものである．

本章では，２節でプラントのモデル化に用いる定性ネットワークモデルを示し，３節でそのモデル記述言語の概要を述べる．さらに，４節ではモデル記述言語の特徴であるオブジェクトの結合と多重継承について，５節で実際のシステムのインプリメントについて述べる．

３．２　定性ネットワークモデル

定性ネットワークモデルではプラントの各状態変数をネットワークのノードで，変数間の作用をアークで表すものとする．各変数は定性値としてその主値と１階からn階までの時間微分値を持つ．今回作成したシステムでは，定性値として次のようなものを用いる．

０	値が０あるいは基準値
＋	値が正あるいは基準値より大
－	値が負あるいは基準値より小
？	値が決定不能

ただし，定性値の形式や，最大階数nの値は定性シミュレーションのアルゴリズムや求められる精度によって異なってくる．また，変数はこのような定性値とは別に，文字列情報として状態値を持つものとする．状態値は故障や運転モードなど，定性値では表現できない情報を扱うのに用いられる．バルブの開度を例とすると，定性値の主値は０のとき通常の開度，＋はより開いた状態，－はより閉じた状態を表す．そして，文字列情報として'close'で完全に閉まった状態を，'stuck'で現在の状態でバルブが固定された故障を表す等の記述が考えられる．

アークとしては，定性値間の物理作用を表すアーク，状態値をからの作用を表すアーク（作用アーク）とそれらの作用の条件を与えるアーク（条件アーク）の３種類を用いる．

物理作用を表すアークには，作用の形式を表す属性を与える．例をあげると次のような属性が与えられる．

１．作用の種類　入力となる変数から出力の変数にどのような作用が行われるかを記述する．例えば，正の作用は＋，負の作用は－など．
２．階数差　　入力の変数から出力の変数への作用の微分階数の差を整数値で与える．

この他にも，シミュレーションではもっと詳細な作用の記述やパラメータがアークに与えられる．

タンクに対する定性ネットワークモデルを物理作用のアークのみで記述した例をFig. 3.1に示す．

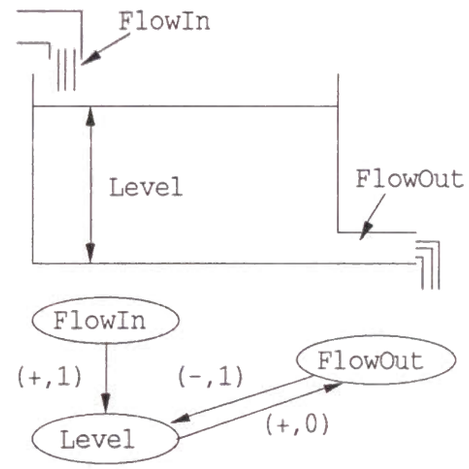

Fig.3.1 Network example of the simple flow

変数はFlowIn（流入量），FlowOut（流出量），Level（タンク水量）の３つが与えられる．変数間の作用はFlowInからLevelへ１階差で正の作用，LevelからFlowOutへ正の作用，FlowOutからLevelへ１階差で負の作用が与えられる．

状態値に関するアークは，定性値と状態値，あるいは状態値間の作用を示すもので，次の３つの種類のアークを用意した．

（１）定性値から状態値への変換を行うアーク
（２）状態値から定性値への変換を行うアーク
（３）状態値間の変換を行うアーク

定性値は主値，微分値のいずれでも参照できる．また，これらのアークには作用が起こるまでの，遅れ時間を指定できる．バルブの例をFig.3.2に示す．

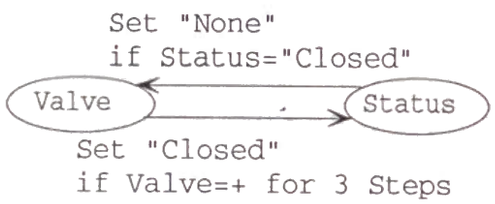

Fig.3.2 Example of the mode control

この例ではバルブの開度を示す変数Valveと，バルブの状態を示す状態変数Statusについて，前述のバルブの動作モード'closed'とバルブの定性値の関係を記述している．

また，条件アークは変数から作用を表すアークへのアークとして記述され，記述された条件により，作用を抑制，あるいは活性化する．条件は定性値，または状態値を参照する形式で記述し，また、条件が成立するまでの遅れ時間を与えることができる．外部からコントロールを受けるバルブに対して'stuck'の故障を記述した例を，Fig.3.3に示す．

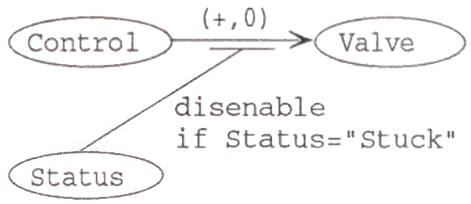

Fig.3.3 Example of the conditional arc

この例では，変数controlが外部からのコントロールを示し，変数 Statusの状態値が'stuck'の時には，コントロールのアークを抑制する．

基本的にはこのような枠組みでフロー構造のモデル化を行なう．次節では，このモデル構築に用いる記述言語について述べる．

３．３　オブジェクト指向アプローチによる記述言語

３．３．１　ネットワークの段階的構築

定性ネットワークモデルの構築の際に，大規模プラント全体のモデルを変数の関係を記述することで一度に組み上げるのは困難である．そのため各プラントのコンポーネント毎にモデルを作成し全体を組み上げたり，プラントの各コンポーネントに対するモデルのライブラリをあらかじめ用意しておき，その部品を用いて全体を構成する方法が考えられる．そのようなモデル記述の枠組みとしてオブジェクト指向アプローチに基づく知識表現を用意した。

プラントなどの物理系を階層的にモデル化する場合は，プラントの各要素の抽象的な記述をクラスとして作成し，実際のプラントの各要素をインスタンスとして作成する．その例をFig.3.4に示す．

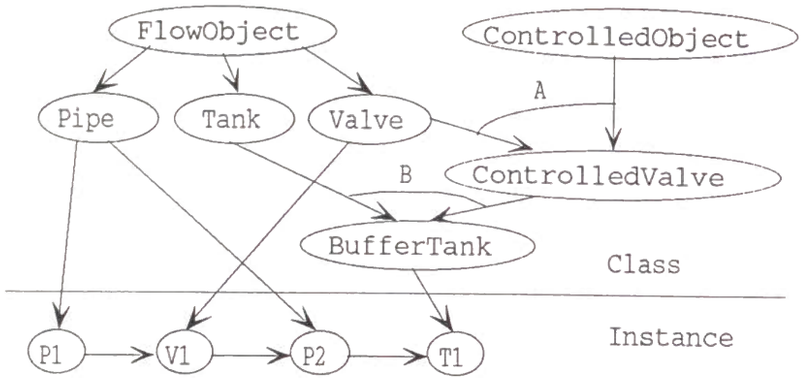

Fig.3.4 Objective explanation of the plant objects

この例では，物流系全体を表すクラスFlowObjectを作成し，そのサブクラスとしてPipe、Tank、Valveなど物流系に拘束や制御を与えるクラスを作成した．また，(a)のようにValveとControlledObjectの性質を多重継承することで，クラスControlled Valveを作成したり，(b)のようにTankとValveを結合しクラスBufferTankを作成するなど，複数クラスからの継承も行なわれる．P1,P2,V1、T1などは実際のプラントのコンポーネントを表すインスタンスで，それぞれの上位クラスから性質を継承している．また，インスタンス間にはそれらの結合を表すリンクが作成され，物流や信号の入出力関係が記述される．

実際にはそれぞれのクラスは，そのクラスに含まれる変数からなるネットワークとして作成される．サブクラスやインスタンスは上位クラスのネットワークを継承し，新たに変数，アークを付け加えることで構成される．また，物理的なフローや情報の流れを表すため，インスタンスはリンクにより結合される．また，継承においても(b)ではクラス間の結合が行われる．今回作成したシステムでは従来の階層的な知識表現に加え，このようなクラス，インスタンス間の結合，および結合を含む継承が特徴となる．

3．3．2　モデル記述言語

次にオブジェクトの記述の概要を示す．


```
class:
  or          name;                                          (1)
instance:
super:
  or          class-name-list                                (2)
component:
variables:
  variable-name: variable-type-list;                         (3)
            :
relations:
 rel-name:
  output-variable<-rel-type:input-variables-list
                  relation-description;                      (4)
            :
in-links:
  class-name-1 - class-name-2(link-type-list);               (5)
            :
-------------------------------------------------------------------------
out-links:
  class-name(link-type-list);                                (6)
            :
                                        instance definition only
-------------------------------------------------------------------------
end;
```

(1)は定義するオブジェクト名の宣言であり，クラスの場合はclass:，インスタンスの

場合はinstance:で定義する．(2)はそのオブジェクトの上位クラスの宣言で多重継承も可能であり，クラス名のリストで与えられ，詳細は次節で述べる．(3)はそのオブジェクトで新たに定義される変数の宣言で，必要に応じて型を付けて宣言する．変数の型の１つとして，オブジェクト間の結合に用いられる結合型（link-type)があり，次のように宣言する．

in(link-type-list)
 link-type-listで与えられる結合型を持つ入力変数
out(link-type-list)
 link-type-listで与えられる結合型を持つ出力変数

実際には結合型はその結合により伝達される信号や，物理量の種類を表すのに用いる．詳しくは，３．４．２　オブジェクトの結合の節で述べる．

(4)は変数間の関係の宣言で，ネットワークにおけるアークを記述する．rel-typeは２節で述べたアークの属性を与えるものである．各アークにはrelation-descriptionに条件を与えることが可能で，実際には前述の条件アークとして展開される．また，relation-descriptionにはそのrelationに対する付加情報を記述できる．これは，変数の型と組み合せて用い，状況に応じて変数やrelationの使用を制限することで，B.Falkenhainer,K.D.Forbus[1988]の方法におけるシミュレーションの視点のような役割をもたせることも考えられる．

(5)はクラス内での結合リンクの定義．(6)はそのインスタンスから出力される結合リンクの定義で詳細は次節以降で述べる．

オブジェクトは，(2)で指定した上位クラスのネットワークに，定義するオブジェクトの差分としての，(3)で定義される変数と(4)で定義されるアークを付け加えることで構成される．このような記述言語により，段階的にネットワークモデルを構築することが可能である．例として，タンクとパイプを記述したものを示す．


```
class: FlowObject;
variables:
   FlowIn: in(flow);
   FlowOut: out(flow);
end;

class: Tank;
super: FlowObject;
variables: Level;
relations:
   Level<-(+,1): FlowIn;
   Level<-(-,1): FlowOut;
   FlowOut<-(+,0): Level;
end;
 end;

class: Pipe;
super: FlowObject;
relations:
   FlowOut<-(+,0):FlowIn;
end;
```

３．４　多重継承とオブジェクトの結合

３．４．１　多重継承

多重継承とは，複数のクラスを上位クラスとする継承の方式である．多重継承を用いることで，単純継承の場合と比較して，オブジェクトに対する記述のモジュール化をより徹底できる．例えば，Ａ，Ｂにおいて，性質αを共有しているとする．また，Ａは他に性質βをＢは性質γを持つものとする．この時，Ａ，Ｂをそれぞれクラスとして定義することを考える．単純継承のシステムでは，αは共有されるので１つのクラスとして定義し，そのサブクラスとしてβを加えクラスＡを，γを加えクラスＢを作成する．この時，β＋γの性質を持つクラスＣを作成するには，Ｃのクラス記述の中でβ，γの記述が新たに必要になり，Ａ，Ｂの記述と重複する部分が発生する．多重継承が可能なシステムでは，α，β，γの記述をそれぞれ独立したクラスとして定義し，それらからの多重継承を行うことで，Ａ，Ｂ，Ｃを簡単に定義でき，α，β，γに対する記述も重複しない．

通常の多重継承では，クラスは上位クラスの性質を受け継いで構成される．本システムで，このような従来の意味での多重継承を行うには，"super:"を用いて上位クラスを指定する．"super:"を用いた多重継承の例を次に示す．

[例４．１]

```
class: ControlledObject;
variables:
 SigIn:in(signal);
end;

class: Valve;
super: FlowObject;
variables:
  Level;
relations:
  FlowOut<-(+,0):FlowIn;
  FlowIn<-(-,0):Level;
end;

class: ControlledValve;
super: Valve,ControlledObject
relations:
  Level<-(+,0): SigIn;
end;
```


この例はFig.3.4のAの部分を記述したもので，ControlledObjectの性質とValveの性質を持つオブジェクトControlledValveを定義している．このような継承の方式はGensymののG2(10)などでも，上位クラスの利用として使われており，オブジェクト指向における継承の方式として一般的なものである．

しかし，プラントのようにコンポーネントの結合により構成されたシステムをこのような枠組で記述する場合，性質の継承だけでなく，クラス自身を組み合わせて新しいクラスを定義できる必要がある．例をあげると，バルブの両端にパイプの接続されたオブジェクトをクラスとして定義する場合，このクラスはパイプ２本とバルブから構成されることになり，"super:"による性質の継承のみでは記述できない．このような，継承を可能にするためには，次のことが必要になる．

（１）上位クラスを構成要素として取り込み，１つのクラスを多重に上位クラスとして

指定できる.

（２）多重継承により指定した複数個のクラスに同じ名前の変数やrelationが存在するとき，それぞれを独立に扱える.

（３）多重継承で指定したクラスの結合が可能である.

一般のオブジェクト指向型のシステムでは，多重継承は可能でも１つのクラスの多重化は不可能であり，同じ名前の変数や手続き（本システムではrelationにあたる）は禁止されたり，１つのみが選択されるなどの処理が行われており，このような継承の方法の実現は困難である．また，（１）（３）のような継承の方式は，オブジェクト指向のＣＡＤシステムで，図形を組み合わせて新しい図形を定義する際に，グラフィカルに図形を組み合わせて新しい図形を定義するのと同じ考え方である．本システムでは，"super:"を用いた従来の継承方式に加え，（１），（２），（３）の条件を満たすコンポーネント継承を導入し，２種類の継承方式を明示的に区別できるようにした．コンポーネント継承では"component:"を用いてその構成クラスを指定し，次のように定義する.

```
＜component-definition＞ ::=
  component: <CompClassList> ;
<CompClassList> ::=
 <CompClass> | <CompClass>,<CompClassList>
<CompClass> ::=
  <ClassName> | <ClassName>[<num>]
  | <NameChangeDescription>
```

まず，（１）に関しては，通常の多重継承では"super:"でクラス名のリストで指定していた上位クラスを，コンポーネント継承では"component:"でクラス名の指定に配列表記を用いることで，同じクラスのオブジェクトが複数含まれるコンポーネントを記述する.

配列表記によるクラスの，コンポーネント内での各オブジェクトの指定は，添字付きのクラス名で行うが，知識の記述時の可読性を高めるために，次のように構成クラスの名前を変更し，その名前で指定することも可能である.

```
<NameChangeDescription> ::=
  <ClassName>[num]={<new-name-list>}
```

また，（３）に関しては構成クラス間の結合リンクを，"in-links:"を用いて次のように定義する.

```
<internal-link-definition> ::=
  in-links: <internal-link-list> ;
<internal-link-list>  ::=
      <internal-link> |
      <internal-link>,<internal-link-list>
  <internal-link> ::=
     <class1> - <class2> ( <link-type-list> )
   | <class1> - <class2>
```

internal-linkはclass1からclass2へのlink-typeの結合型を持つ結合リンクを表し，class1,class2はともにcomponentで指定された要素のクラスである．実際の結合は通常のオブジェクト間の結合と同様に行われる．例として，バルブの両端にパイプの結合したクラスPipeValveを例として示す.

[例４．２]

```
class: PipeValve;
component: Pipe[2]={In,Out},Valve;
in-links: In-Valve(flow),Valve-Out(flow);
end;
```

この例では，パイプ２本を"In","Out"と名前を付けて，"In"から"Valve"，"Valve"から"Out"に結合している．

（２）に関しては，　"component:"で指定されたクラスの変数名を，構成クラス名と組み合わせて次のように変換し，他の構成クラスに同じ名前の変数が存在しても一意に決定されるようにする．

　　component-class-name/variable-name

[例４．２]では次に示す変数が生成される

original class	generated variables
Pipe[1]	In/FlowIn,In/FlowOut
Valve	Valve/FlowIn,Valve/Level, Valve/FlowOut
Pipe[2]	Out/FlowIn,Out/FlowOut

コンポーネントが多重化した場合，変数名には同様に"/"で区切られたクラス名が連結される．それに対して，super:による多重継承では，変数名の変更は行われない．また，同じ名前の変数が複数存在する場合は，最も下位のクラス階層に含まれるものが，super:の上位クラスのリストにおける記述順に探索され，最初に見つかったものが使用される．

コンポーネント継承は，オブジェクト指向における部品の細分化，モジュール化の考え方からは，逆行するように考えられる．しかし，プラント等のような部品の組合わせからなるシステムをモデル化するには，オブジェクトの結合と組み合わせて使うことにより，非常に有用な方法となる．また，後述のようにコンポーネント継承はインスタンスの生成時に処理されており，コンポーネント継承で生成されたインスタンスに対しても，システム内ではモジュール化されている．

３．４．２　オブジェクトの結合

このシステムでは，クラス，インスタンス単位でネットワークを作成し，それを結合することで全体のネットワークを作成する．しかし，実際に定性ネットワーク上で結合されるのは変数であるため，オブジェクト間の結合を変数間の結合にマッピングする必要がある．１つの方法として各オブジェクトにおいて，結合する変数を直接に指定するようにすると，あるオブジェクトの知識を記述する際に，結合される側のオブジェクトに関する情報が必要になり，オブジェクトの独立性が失われる．そこで，変数と結合リンクに結合型をリストとして与え，そのマッチングにより間接的に結合を指定する方法を用いる．プラントのような物理系を表す定性ネットワークでは，各種のフローや信号など各変数の持つ情報の種類は，各オブジェクトにおいて共通と考えられるため，その情報の種類を結合型として用いることができる．

結合型の記述は次のようになる．

```
< link-type-list > ::=
        empty | <link-description>
        | <link-description>,<link-type-list>
< link-description > ::= <link-type>                    (1)
            | <link-type-1> = <link-type-2>             (2)
```

実際には，次のような規則に基づいて結合される．結合リンクが結合型のリスト(link-type-list)を持たない場合は，リンクの入力側のオブジェクトの出力変数とリンクの出力側の変数の入力変数から，結合型の一致するものを全て結合する．物理的に見ると，これは結合される２つのオブジェクト間で，共有される全ての情報が結合されることになる．しかし，この方法では双方向に種類の違うフローが存在する場合などが表現できない．そのために，リンクに与える結合型で結合される情報を選択できるようにした．つまり、(1)の場合は，link-typeで与えられた結合型それぞれに対して，変数の結合型と一致するもののみが結合される．(2)の場合はそのオブジェクトのlink-type-1の結合型の変数と，結合される側のオブジェクトのlink-type-2の結合型の変数が結合される．結合型の指定は，オブジェクト間の結合を行う要素を指定することになり，例えば，ＡとＢは電気的に結合されているという場合を表現することになる．リンクの結合型を用いた例を次に示す．

[例４．４]

```
class:SigTank;
super: Tank;
variables: SigOut(signal);
relation:
 SigOut<-(+,0):Level;
end;

instance:v1;
super:ControlledValve;
out-links: t1(Flow);
end;
```



```
instance:t1;
super:SigTank;
out-links:v1(signal),p1(flow);
end;

instance:p1;
super:Pipe;
end;
```

この例ではタンクt1からの出力が，パイプp1には物流関係として，バルブにはコントロール信号として結合される．また，前章で述べたコンポーネント内リンクについては外部のリンクと同様に結合が行われるが，コンポーネント内リンクで結合に用いられた変数は，外部リンクの結合には利用されない．（外部から見ると結合型の属性がなくなる．）このような結合型を用いた方法により，オブジェクト毎に記述の独立性を保ち，かつ柔軟な結合の記述が可能とした．

ただし，同じ結合型の入力や出力を２つ以上もつようなオブジェクト（例えば同じ種類のフロー出力が２つ存在する場合）に対しては，その結合型ではどちらに結合するかが区別できなくなる．そのため，物理的な性質を表す結合型のみでなく，新たに結合型を（MainFlow,SubFlowのように）加え，結合型の記述(2)の方法で結合型の変更を行う必要がある．これは下位クラスにおいて上位クラスの型を詳細化した型階層[11]を与えることに相当する．物理的にみると，同じ型(Flow)をを持った変数に対し，その構造的な位置関係（右，左など）やその物流の性質(Main,Subなど）のような分別に用いることが可能な型を加えることになる．また，シミュレーション時に結合部における作用を双方向に同等とすると，型階層を持つオブジェクト間の結合においては，下位の型階層を持つオブジェクトから上位の型階層のオブジェクトに結合を行ったほうが好ましい．これは，下位の階層からは上位の階層の結合型は透明であり結合型を決定できるがが，

逆の場合は不透明であるからである．

３．４．３　モデルの記述例

この節では簡単な例を用いて，コンポーネント継承の有用性を示す．ここでは，バッファータンクをクラスとして定義する場合について考える．[例４．４]では，１つ１つをインスタンスとして定義しそれを結合して全体を作成したが，ここでは全体を１つのクラスとして定義する．従来の多重継承のみ用いる場合，SigTank,Pipe,ControlledValeveを多重継承するとFlowIn,FlowOut,Level1などの変数が重複することになり，不都合が生じる．そこで中心のオブジェクトとなるSigTankを上位クラスとして残りの変数を付け加えることで構成すると次のようになる．

```
class:BufferTank
super:SigTank;
variables: VFlowIn:in(flow),VFlowOut,VLevel,
           VSigIn,PFlowIn,PFlowOut:out(flow);          (1)
relations:
     VFlowOut<-(+,0):VFlowIn;
     VFlowOut<-(-,0):VLevel;
     VLevel<-(+,0):VSigIn;                           (2)

     PFlowOut<-(+,0):PFlowIn;                        (3)

     FlowIn<-(+,0):VFlowOut;
     VSigIn<-(+,0):SigOut;
     PFlowIn<-(+,0):FlowOut;                         (4)
end;
```


この場合(1)の変数の記述，(2),(3)のrelationの記述がControlledValve，Pipeのクラス定義とシステム内で重複する．また，(4)のように結合部に関してもrelationで記述する必要がある．また多重継承時の変数の重複による障害を防ぐメカニズムが用意されていたとし，Tank,ControlledValve,Pipeからの多重継承によりBufferTankを構成したとしても，(4)の結合に関する記述は変数を指定してrelationで記述することになる．

そこで，コンポーネント継承と内部リンクによる記述を用いた場合は次のようになる．

```
class:BufferTank;
component:SigTank,ControlledValve,Pipe;
in-links: ControlledValve-Tank(flow),
          Tank-ControlledValve(signal),
          Tank-Pipe(flow);
end;
```

このように，３つのクラスをコンポーネントとして指定し，外部リンクにより結合する場合と同様に内部リンクとして結合を記述することで目的のクラスが得られ，直感的にも理解しやすい．このような機能は，オブジェクトの組み合せが多数存在するほど効果が高くなる．例えばこのようなバッファタンクを直列,および並列に接続した対象に対するクラスを作成するには，次のように記述すればよい．

直列の場合

```
class:SirialBufferTank;
component:BufferTank[2]={Front,Back};
in-links:Front-Back;
end;
```

並列の場合

```
class:PallarelBufferTank;
component:BufferTank[2]={Tank1,Tank2},
          Pipe[2]={In,Out};
in-links: In-Tank1,In-Tank2,Tank1-Out,Tank2-Out;
end;
```

このようにコンポーネント継承とコンポーネント内リンクによる記述は，物理系のように構造を持つオブジェクトの定義を行う場合に次のような効果がある．

（1）オブジェクトの内部記述の重複を避けることができるため，記述量が削減される．
（2）オブジェクトが，その部品となるオブジェクトの組み合せにより表現できるため，理解し易い記述が可能であり，また，オブジェクト内部の部品の組み替えも容易にできる．

3．5　オブジェクト記述の実現

3．5．1　ネットワークの実現

これまで述べてきたように，知識の構築はクラス単位で段階的に行われる．しかし，シミュレーションの実行は，インスタンスを結合した定性ネットワークとして行われる．シミュレーションの実行時には，各変数において，アークの入力側の変数と出力側の変数の値の参照，変更が頻繁に行われる．ここで，変数間の結合関係を，アクセスの度に上位クラスから参照すると，非常に時間を要する．また，実行前にオブジェクトのコンパイルを行い，アークを変数に対して完全に割り当てることでこの問題は避けられる．しかし，この方法ではあるクラスの知識を書き換える度に，そのサブクラスは全てコンパイルしなおす必要があり，知識のメンテナンスの点で問題がある．

そのために本システムでは，中間的にコンパイルすることで，実行時の能率を保ち，メンテナンスに関してもかなりの融通性を持つ方法を用いた．

クラス間の階層関係はポインタを用いて参照し，ネットワークの生成時に，各インスタンスに含まれる変数やアークを生成する．変数に関しては，上位クラスで定義された変数も含め，定性値，状態値などの各インスタンス独自の情報は各インスタンス内に持たせる．ただし，変数の持つ各属性はその変数の定義されたクラスで管理し，インスタンス内の変数からはポインタにより参照する．アークに対しても同様に，各インスタンスの変数間に展開し，その属性はアークの定義されたクラスで管理し，アークからはポインタにより参照する．その概念図をFig.3.5に示す．

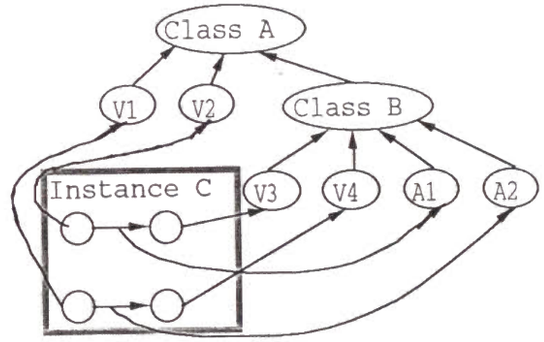

Fig.3.5 Generation of the network

このような実現法により，上位クラスにおける変数，アークの各属性の書き換え時にインスタンスをコンパイルしなおす必要がなくなる．ただし，この方法では変数，アークの付加や削除の場合は再コンパイルしなければならない場合がある．しかし，実際の知識のメンテナンスでは各属性の調整が主であり，あまり問題にならないと考えられる．また，このような属性値の変更はシステムに組み込まれたクラスブラウザーにより行われる．

また，コンポーネント継承を含むインスタンスには次のようなアルゴリズムで生成される．

1.コンポーネントの構成要素のクラスに属するインスタンスをダミーの名(class1,class2など)を付けて生成する．（インスタンスの生成時には，上に述べた方法で変数が生成され，relationがアークとして展開される．）ただし，配列表現のクラスの場合，その個数分だけ生成する．
（この際にコンポーネント継承がネストしている場合は，再帰的にアルゴリズムが実行される．）
2.生成したインスタンスのダミー名と元のクラス名の対応表を使って，クラス内リンクをインスタンス間に生成する．
3.各ダミーのインスタンスの変数名に，そのクラス名を結合し変数名を変更し，目的のインスタンスに移す．このときに変数を移すことで同時にアークも移される．全ての変数を移した後，ダミーのインスタンスは消去する．

このように，既存のインスタンス生成のメカニズムを用いて，コンポーネント継承を実現できる．また，コンポーネントの構成要素のそれぞれが，その要素を表す上位クラスのインスタンスとして生成されることになる。つまり、コンポーネント継承はモデルの記述上でのインスタンスのブロック化として働くことになり，システム内でのクラス間のモジュール化を疎外することはない．

3．5．2　システムの概要

今回作成した，システムの概要をFig.3.6に示す．各ユニットの働きを述べると，まず，コマンドインタプリタはユーザーの入力したコマンドを解析し，各ユニットを呼び出し，また、バッチ的な処理を行ったり，システムの動作モードの設定，シミュレーションの初期状態の設定，出力先の切り替えなどを行う．コンパイラでは，前述の記述言語で書かれたプラントの構造的知識を定性ネットワークに変換する．シミュレータは与

えられた初期状態に基づき，定性シミュレーションを行う．ブラウザーでは，現在入力されている定性ネットワークに対し，各変数やアークの持つ属性の変更，アークの追加，削除，新しいオブジェクトの追加，等を行い，知識のメンテナンスを行う．

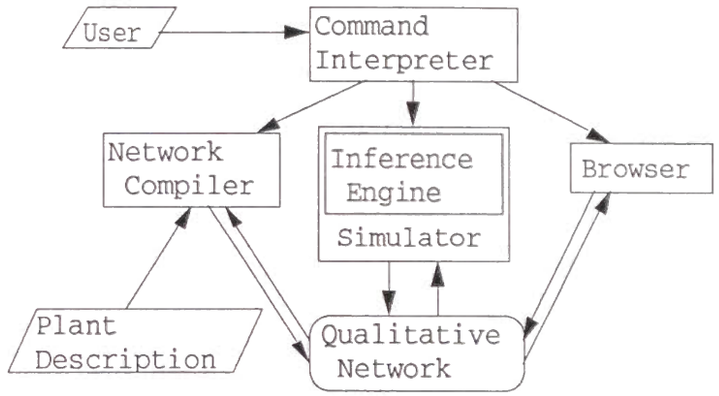

Fig.3.6 Structure of the System

3．6　結言

今回作成したプラントの診断ベース構築システムのための，その定性モデルの知識構築の方法について述べた．その特徴としては次のようなものがある．

(1)プラントの構造的知識をオブジェクト指向アプローチによる知識表現で段階的に構築し，ライブラリィ化が可能である．
(2)プラントのような多くのコンポーネントからなる複雑な物理系の知識表現のために，従来のオブジェクト指向知識表現の継承に加え，構造的な継承のメカニズムを持つ．
(3)リンクタイプを用いた結合方式により，柔軟性のあるオブジェクトの結合が可能である．

しかし，実際にプラントの構造を記述すると，オブジェクトごとに完全に独立して記

述するのが難しい知識も存在する点が問題になる．実際に，プラントのエキスパートが行っている定性的な伝播の予測は，プラント全体のバランスを見て行っており，オブジェクト毎の知識として記述するのは困難である．現在のところ，オブジェクト間の陰蔽性をおとし，オブジェクトから他のオブジェクトの変数を参照することで記述している．しかし，この方法では根本的な解決にはならず，システム全体に対するヒューリスティクスを記述するような枠組が必要であると考えられる．

次章では、本章で定義した定性ネットワーク上での定性シミュレーションのアルゴリズムについて述べる．

第4章　定性ネットワークモデル上の定性シミュレーション

4．1　緒言

本章では，前章で述べた物理系の構造に合わせて，容易に作成できる定性ネットワークをモデルとして用い，その構成とネットワーク上での定性シミュレーションについて考察した．本章で扱う定性ネットワークは，主に線形のシステムを対象とし，部分的に非線形のシステムも扱うものとする．

本章で述べる方法は，de Kleer[5) 20)]の定性シミュレーションの方法を定性ネットワーク上のシミュレーションのために改良したものである．具体的には，定性ネットワークでモデルを表現し，その上で値伝播を行うことでシミュレーションを進めていく．ここで，値伝播はある変数における時間的な値変化を表す動的伝播と，その変化にともなう他の変数の値変化をあらわす静的伝播に分類される。このことにより、時間的な状態変化をおこす動的伝播の選択の方法が簡単になる．また，非線形の伝播に関しては，決定可能な値のみを伝播させることで部分的に表現し，値の収束などの特殊な場合のみ検出するものとする．また，Kuipers[4)]の定性シミュレーションの方法では，本章の方法と同じくネットワーク上での伝播を用いシミュレーションを行うものであるが，モデル構築の手法，シミュレーションの方法などの点で異なっている．また、前章で述べたように，知識記述においてはオブジェクト指向アプローチに基づき各部分ごとにモデルを作成し，それを結合させて全体を構成する枠組みを用意している．そのために変数の結合時の伝播規則に対する考察も行った．

本章では２節で定性ネットワークについてシミュレーションの観点から述べ，３節でそのシミュレーションの方法の概略を述べる．そして，４節ではネットワークの一貫性まで考えたシミュレーションについて述べ，５節でdeKeer,Kuipersの方法との違いを述べ，６節では例題を示す．

4．2　定性ネットワークモデル

前章でモデルの記述の観点から定性ネットワークモデルについて述べたが，ここでは定性シミュレーションの観点から述べる．定性ネットワークモデルは、対象となるシステムの各状態変数をネットワークのノードで，変数間の物理的な関係をアークで表すものである．ただし、全ての変数はある一点の値を基準値として持ち、全ての変数が基準値にあるとき、またその時に限り、そのシステムは定常状態にあるものとする。今回のシミュレーションには主値と１階から４階までの微分値の定性値を用い，３階の値までを表示する．変数の値は，主値，１階微分値，２階微分値、３階微分値の順に並べたベクトル形式で，次のように表記する．

FlowIn=(0,+,-,+)

ただし、0は値が0あるいは基準値として１点を表すものとし、ある領域を持つようなものは扱わない。また，前章で述べたように物理作用を与えるアークには，各種の属性を与える．ここでは本章の定性シミュレーションで用いるもののみを列挙しておく．

１．作用の種類

①線形に正の作用　+　　線形に負の作用　-

②値が等しい　　=

③値が0のときのみ0を伝播し，その他の場合は決定不能　　?

④値の絶対値（負の絶対値）を伝播　　++（--）

③，④は線形の作用（+，-）のみでは表現できない非線形の作用を表すのに用いる．

２．階数差　アークの入力の変数から出力の変数への作用の微分値の階数の差をを整数値で与える．ただし，本論文のシミュレーションに用いているシステムでは変数の微分階数が４階までなので，余り大きな数は意味がなく，２階差ぐらいまでが適当である．

また，作用アークは、変数の全ての階数に対して共通に与える場合と，各微分階数に別々に与える場合がある。基本的には、線形の作用の場合は全ての階数間で共通とし、非線形の場合は別々に与える。これは、定性方程式において、線形のシステムを表す式は微分を繰り返しても、変数間の関係および微分階数の差は変化しないが、非線形の場合には、式の変形が起こり階数差のみでは関係が定義できないためである。ここで、Fig.3.1のバッファータンクの例に対して、詳細なモデルを作成した例を示す。Fig.3.1のモデルではLevelとFlowOutの関係を単純に線形の正の作用としているが、実際には次の式で表される非線形の作用である。

$$FlowOut = a\cdot(Level + b)^{1/2}$$

この式には、非線型要素が含まれるため、微分値に関しては別に関係を求める必要がある。

$$dFlowOut/dt = a/2\cdot(Level+b)^{-1/2}\cdot dLevel/dt$$

$$d^2FlowOut/dt^2 = -a/4\cdot(level+b)^{-3/2}\cdot(dLevel/dt)^2 + a/2\cdot(Level+b)^{-1/2}\cdot d^2level/dt^2$$

(Level+b)の符号変化はないので、１階微分値間は正の作用、FlowOutの２階微分値へはLevelの１階微分値から絶対値で負の作用、Levelの２階微分値からは正の作用となり、Fig.4.1のモデルが得られる。このモデルは、この系に対する次の方程式から作成したものと一致する。

$$dLevel/dt = c\cdot(FlowIn-FlowOut)$$

$$FlowOut = a\cdot(Level + b)^{1/2}$$

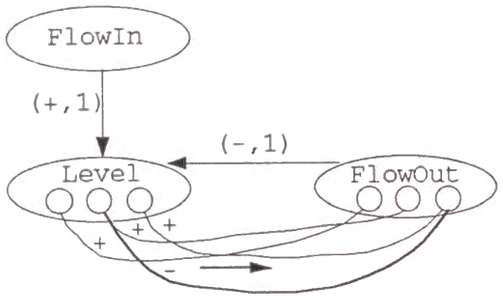

Fig.4.1 Modified network model for buffer tank

また、実際に作成したシステムでは、定性値に関しては大きさのオーダー付けによる表現、領域を持つ基準値の処理、作用アークに関しては、作用の優先度、作用の大きさのオーダー付け、作用の伝播の遅れ時間の表現などを含むものであるが、本論文では触れない。

ここで、定性ネットワークのモデル化対象に対する条件をまとめておく。(1)基本的には線形のシステムであり、一部にのみ非線形の部分を許す、(2)モデルは変数間の作用をアークで表したものを用いる、(3)各変数は１点の値を基準値として持ち、全ての変数が基準値の時、かつその時に限りシステムは定常状態にある。

このような，定性ネットワークを用いて，プラントなどの物理系を記述する．ただし，実際には前章で述べたような記述言語を用いてモデルの記述が行われる．そのシミュレーションの方法について次節以降で述べる．

４．３　定性シミュレーション

４．３．１　基本伝播規則

定性ネットワークに対して，ある変数に外乱を加え，その値を伝播させることで定性シミュレーションが行われる．定性ネットワーク上の値伝播は，値をアークにより伝播させる静的伝播と，変数内部における値の動的な変化を扱う動的伝播に伝播を分けられる．動的伝播は系の時間変化を発生させるもので、実際にはいずれかの微分値をその一階下の値の変化方向として伝播を行い、変化に時間を要するものと瞬時に発生するものがある。静的伝播は、初期値の設定や動的伝播の発生に伴う変数（微分値も含む）の値変化を、他の変数に伝播させるもので、実際にはある変数からアークで結合された変数（自分自身の場合もある）に対し、同じ階数の値あるいはより高階の微分値への伝播として処理される。また、静的伝播はアークで結合された複数の変数およびその微分値の同時変化を表すものであり、発生の元になる変化と同時に発生する。全体的には、動的伝播が状態変化を起こす引金となり、静的伝播でネットワーク全体のバランスをとるということになる。定性シミュレーションは、動的伝播とそれに伴う静的伝播、および、それらの値変化に伴う動的伝播の検出を繰返し実行することで、進められる。次にそれらの実行について述べる。

(1) 静的伝播

静的伝播はある変数における定性値の変化をアークによって他の変数に伝播させる作用で，次の静的伝播規則により実行される．

(a) 静的伝播規則

ある変数においてn階(n>=0)の値が変化したとき、次の静的伝播規則によりその値の伝播を行う。

(a-1) 作用が+のアークに対しては，その値をそのまま出力側の変数へ伝播し，作用方向が-の時は，符号を反転して伝播する．作用が++(--)のアークに対しては，その入力の(負の) 絶対値の伝播を行う．作用が?のアークに対しては，値０の時のみ０をそうでない場合は?を伝播する．ただし，値?を伝播する場合はどのようなアークに対しても?が伝播される．

(a-2)階数差は，階数差の分だけ伝播させる変数の階数を増やして伝播する．つまり、階数差がmの時は、出力側の変数のn+m階の値に伝播される。

(a-3)変数への入力アークが１つのみのときは，そのアークにより伝播された値が，そのまま変数値になる．変数が複数の入力アークを持つときは，その値は全てのアークにより伝播される値の和となる．定性値の和の値は，Table.4.1に基づく定性演算により決定される．

A　　B	－	0	＋	?
－	－	－	?	?
0	－	0	＋	?
＋	?	＋	＋	?
?	?	?	?	?

Table 4.1 Qualitative addition A+B

(2) 動的伝播

微分値はその１つ低階の値の変化量になるため，変化が継続することでその値が定量

値におけるしきい値に達し，定性値に変化を起こす．例えば、V=(+、-、0)の場合，１階の微分値'-'が主値（0階）の'+'に減少の作用をし，V=(0、-、0)となり，さらにV=(-、-、0)と変化する．このように微分値から，その低階の値に対する作用が動的伝播である．動的伝播は次の動的伝播規則により処理される．

(b)動的伝播規則

(b-1) 定性値が+,-の微分値は，１つ低階の値に同じ方向の作用を行う．その作用結果はTable4.2に従う．

(b-2) 動的伝播の際に変化した値は，変化直後に静的伝播も行ない，他の変数を変化させる．

X　　dx	－	＋	?
－	－	0	?
0	－	＋	?
＋	0	＋	?

Table 4.2 Dynamic propagation

ただし、微分値の値が0以外の値であっても、必ずしも次のステップでTable.4.2に定義された動的伝播が発生するとは限らず、次ステップで、必ずTable.4.2で示す値になるという意味ではない。これは、変化に数ステップ要する場合もあり、さらに、後に述べるネットワークの一貫性などの条件により、発生可能な動的伝播が限定されるからであ

る。(b-2)の処理により，結合された複数の変数において同時に発生すべき値変化を１つの動的伝播からの静的伝播により実行する．

また，定性値+,-,0の中で0のみが１点を表すという，特異性から次の規則が導かれる．

(c)0の瞬時性(instant change rule)

定性値0へ動的伝播は，同じ微分値からの作用により，定性値0は瞬時で通過する．つまり，定性値0は１点の値を表すため１つ高階の微分値が0でない場合は，その作用により0から移動することになる．

実際には，0を静的伝播させた直後に動的伝播を繰り返し，0点を通過させる．ただし(b-2)を用いて，0の静的伝播により微分値が変化し，０からの動的伝播が発生しない場合もある．また，この規則はde Kleerのinstant change ruleとDerivative instant change ruleを合わせたものであり，本章の定性ネットワークでは微分値と主値を区別しないため１つの規則とする．

de Kleerの方法では微分値が0でない全ての変数について動的伝播が起こりうるとしているため，多数の動的伝播が発生しうる場合に，その選択の方法が問題になる．本性で述べるアルゴリズムでは次に述べる規則により，独立に発生可能な動的伝播を決定し，それに従属する動的伝播は静的伝播で処理するものとする．

(d) 動的伝播発生の決定則

次の(1),(2),(3)の3つの場合に，独立な動的伝播が発生する．

(1)静的伝播によりフローの合流点において，微分値が変化した場合

(2)0階の値からの静的伝播により微分値が変化した場合

(3)動的伝播により微分値が変化した場合

これらの規則は最も基本となる伝播規則であるが，これらのみでは静的伝播における+,-の競合が多数発生し決定不能の変数が多くなってしまう．例としてFig.4.1のバッファータンクのシミュレーションを行った例を示す．ただし，初期状態としてはFlowIn=(+,0,0)を与えたとする．

step1			rule	
FlowIn	(+,0,0)			(1)
Level	(0,+,0)		(a)(d)	(2)
	Set	Level(0)-DP		(3)
FlowOut	(0,+,-)		(a)	(4)
Level	(0,+,-)		(a)(d)	(5)
	Set	Level(1)-DP		(6)
FlowOut	(0,+,-)		(a)	(7)
Level	(+,+,-)	Execute(3)	(c)(b)	(8)
FlowOut	(+,+,-)		(a)	(9)
Level	(+,?,-)		(a)	(10)
	Set	Level(0)-DP		(11)
FlowOut	(+,?,-)		(a)	(12)
Level	(+,?,?)		(a)	(13)
FlowOut	(+,?,?)		(a)	(14)
step2				
Level	(?,?,?)	Execute(11)	(b)	(15)
FlowOut	(?,?,?)		(a)	(16)

ただし、DPは動的伝播、ruleの欄は値変化あるいは動的伝播の発生に用いられたルールが示されている。このシミュレーションでは，(10)の１階の値に生じた?は+と-の競合

によるものである．このような競合による?の発生を減少させるために，次節にのべる動的バランス則を付け加える．

４．３．２　動的バランス則

動的バランス則は，動的伝播発生の決定則の拡張で，変数に入力される定性値の大きさのバランスを考えて，動的伝播を発生させるものである．動的伝播にともなう静的伝播により?が発生した状態の例ををFig.4.2に示す．

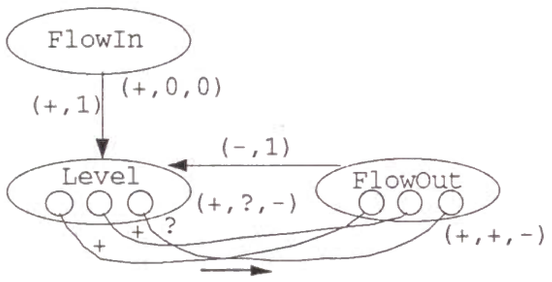

Fig.4.2 Conflict of the values

ここで，FlowInの0階の+の作用と，FlowOutの0階の+の負の作用がLevelの１階への作用で競合し，?が発生する．しかし，FlowOutの+は１階の+からの動的伝播により発生したものとすると，0点を通過した直後の+であり，定量的な大きさは小さいはずである。さらに増加を続けることでFlowInの+とバランスするはずである．つまり、この値をLevelに伝播させると，Levelの１階の値はFlowInの+との大小関係により，+，0，-と変化するはずである．このような値変化を実現するのが動的バランス則である．動的バランス則は動的伝播により引き起こされた変数値の変化の静的伝播において用いられる．

(e) 動的バランス則

動的伝播の結果の静的伝播を行いある変数のn階の変数値(n >= 0)において，+と-の入力が競合するとき（?になるとき），次の処理を行う．

(1)競合する+,-がともに、このステップで発生した動的伝播によるものであれば決定不能(?)とする。

(2) nが打ち切り階数のときは，変数値を変化させずにそのまま終了する．

(3) 同じ変数に対しn+1階の値を決定する．

(4) n階の変数値に対し，(2)で決定されたn+1階の変数値によって，動的伝播をセットする．

(1)の場合はどちらの値も変化直後の値であるため、定性的には値を決定できず、動的バランス則は働かない。また，(3)において，n+1階の値を決定する際に再帰的に呼び出しが発生することもある。再帰的に実行された場合はn+1階とn階の値のどちらを先にバランスさせるかで実行結果は異なってくる．実際には，ここでどちらかを選んで実行することになるが、後に述べるネットワークの一貫性により実行順序が一意に定まる場合もある。

これまでの規則を用いた定性シミュレーションの実行のアルゴリズムをFig.4.3に示す。また、Fig.4.1のネットワークに対する実行例を次に示す．ただし，初期条件としては前節と同じく，FlowIn=(+,0,0)を与えた．

step1			rule	
FlowIn	(+,0,0)			(1)
Level	(0,+,0)		(a)(d)	(2)
	Set	Level(0)-DP		(3)
FlowOut	(0,+,-)		(a)	(4)
Level	(0,+,-)		(a)(d)	(5)

	Set	Level(1)-DP		(6)
FlowOut	(0,+,-)		(a)	(7)
Level	(+,+,-)	Execute(3)	(c)(b)	(8)
FlowOut	(+,+,-)		(a)	(9)
Level	(+,+,-)		(a),(e)	(10)
	Set	Level(1)-Balance-DP		(11)
step2				
Level	(+,0,-)	Execute(11)	(b)	(12)
FlowOut	(+,0,-)		(a)	(13)
Level	(+,0,0)		(a)	(14)
FlowOut	(+,0,0)		(a)	(15)

ただし、Balanceは動的バランスの発生を表す。このシミュレーションの結果は，２階の値までみても実際の値変化に適合している．また，この例は平方根という非線型要素を含むものであるが、非線型要素が含まれる場合でも，このように収束を行うものなどは検出できる場合がある．今回扱っている感度解析としての定性シミュレーションでは外乱が収束するかの判定が重要であり，収束の検出が可能なことは有用である．

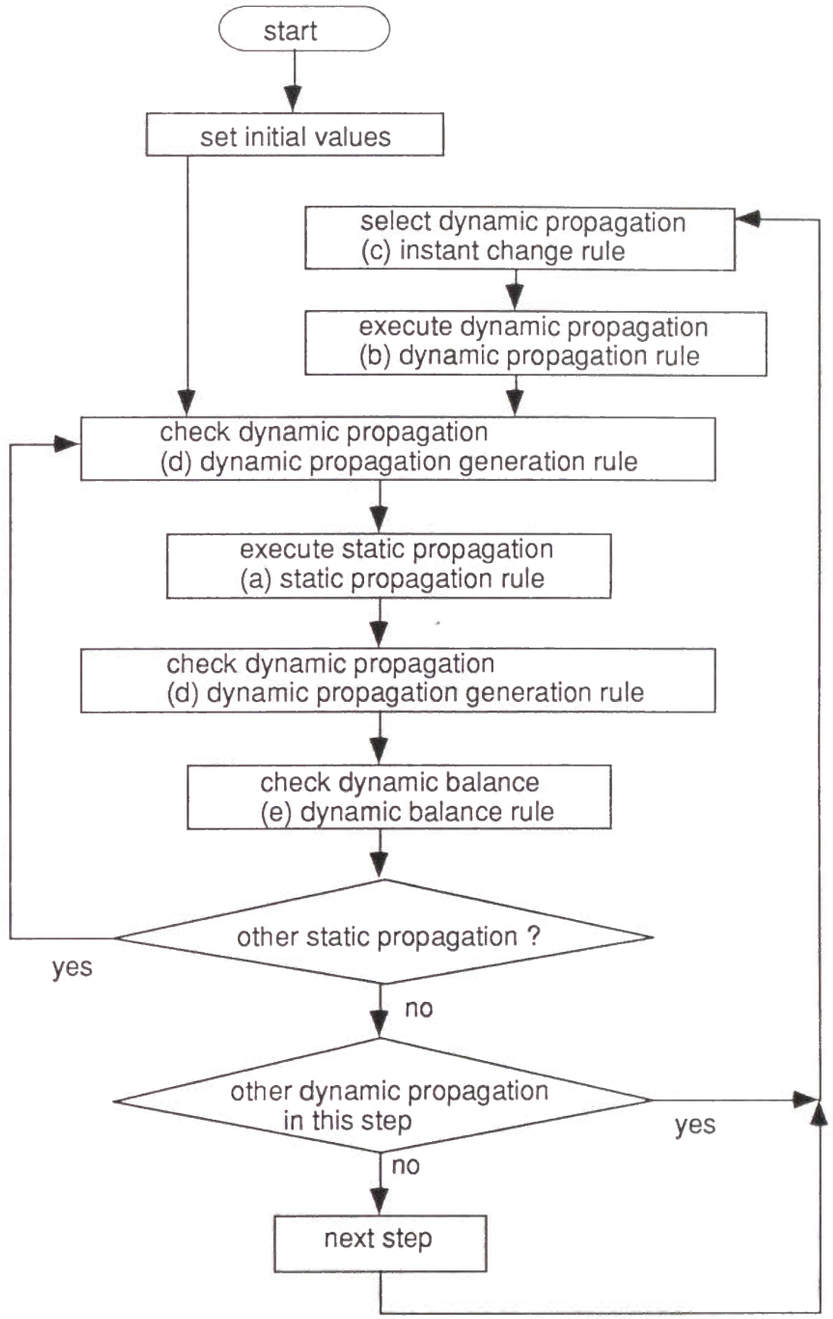

Fig.4.3 Qualitative Simulation Algolithm

４．４　一貫性に基づく伝播の制限

これまで述べてきた規則のみでシミュレーションを実行すると，物理的に起こりえないパスが数多く生成される．これは，伝播を定性ネットワーク上の定性値の一貫性を考えずに実行しているためである．この章では定性値の一貫性まで考慮した伝播の実行について述べる．

４．４．１　入力の一貫性

まず，定性ネットワーク上の各変数における入力の一貫性を次のように定義する．

Def.入力の一貫性

n階以下の階数差の入力アークを持つ変数のn階の値において，全ての入力の定性値としての和が，その変数の定性値となりうる時，n階の入力の一貫性が満たされるとする．

定性値の和は，基本的にはtable1により決定されるが，+，-の両方を含む和に対しては必ずしも?である必要はなく，+，-，0，?のいずれの値でもよい．また，n階の変数値に対して，n+1階以上の階数差の入力アークが存在するときは，そのアークからの入力は評価できないため，一貫性は定義されない．たとえば２階の入力アークをもつ変数においては１階と0階の入力の一貫性は定義されない．

入力の一貫性から，次の規則が導かれる．

(f) 伝播の一貫性　ある変数のn階の値において伝播(動的伝播，静的伝播）が発生するには，次の条件のいずれかが満たされる必要がある．

(1)注目する伝播によるn階の変数値の変化が，その変数のn階の入力の一貫性と，その変数から出力されるアークに結合された変数における入力の一貫性を満たしている．

(2)注目する伝播によるn階の変数値の変化を静的伝播することにより，ネットワーク全体で変数の一貫性が満たされる．（ただし，(1),(2)とも一貫性が定義される階数のみで評価する．）

(1)が満たされる場合は，その変数における伝播の結果を静的伝播させることなしに一貫性が満たされる状態である．

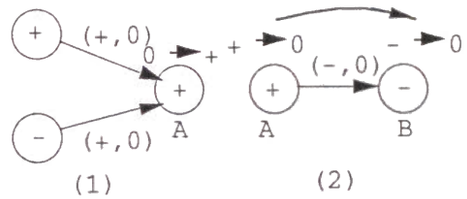

Fig.4.4 Consistency of the propagation

Fig.4.4(1)ではAの０階の値が０から+に変化しても一貫性は満たされている．それに対して，(2)の場合は複数の変数が同時に変化することで，全体の一貫性が満たされる場合であり，それを静的伝播により実現する．Fig.4.4(2)ではAの値が+から0に変化するとBの入力の一貫性が満たされなくなる．そのため変化後の値+を静的伝播させることによりBの値も変化させる．de Kleer(3)の方法では，最初に定性方程式上で矛盾しない全ての定性値の組を求め，その間での状態遷移を推論するものであり，伝播過程での矛盾状態は許されていない．本章で述べる方法では，伝播の過程では矛盾状態を許し，静的伝播により矛盾を解消するという方法を用いることで，定性値の一貫性を調べながら，前向きにシミュレーションを進めて行く．

伝播の一貫性は方程式上での定性値の無矛盾性を保証するためのものであるが，さらにその変化が物理的に可能なものであることが要求される．まず，複数の変数で同時に発生する動的伝播を静的伝播を用いて表すため，その静的伝播に対しては次のことが要求される．

(g) 静的伝播の無矛盾性

動的伝播に伴うn階の変数値への静的伝播において，その変化はその変数自身のn+1階の変数値からの動的伝播により可能でなければならない.

静的伝播の無矛盾性を定性ネットワーク上の条件に置き換えると，次のようになる.

(g-1)変化方向の同一性

動的伝播により発生した，変数Aにおけるn階の値の変化を，アークで結合された変数Bのm階の値に静的伝播させるには次の条件が成り立たなければならない。

(1)アークの作用が+の時

Aのn+1階の値がBのm+1階の値に等しい.

(2)アークの作用が-の時

Aのn+1階の値とBのm+1階の値の符号が互いに逆.

(3)アークの作用が?の時

Bのm+1階の値が0以外

(4)アークの作用が++(--)の時

Aのn+1階の値の絶対値(負の絶対値)がBのm+1階の値に等しい.

((1)～(4)を変化方向の同一性と呼ぶ.)

または，変数Bのm+1階の値が?である.

(g-1)は，アークの作用まで含めて変数間で作用の方向が同じならば静的伝播が可能という静的伝播の条件を与えている. Fig.4.5では，この規則を用いてAの変化を伝播可能な変数とそうでないものを示している.

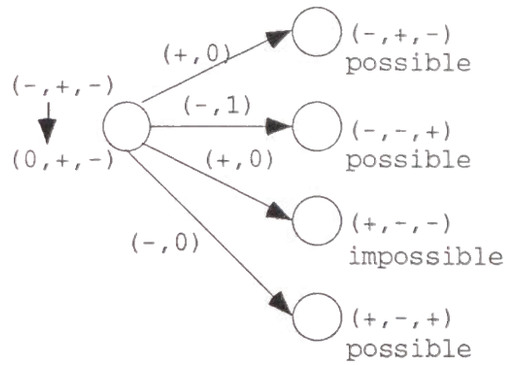

Fig.4.5 Possibility for static propagation

ただし，mが打ち切り階数の場合は，変化方向を決定不能なため，いつでも伝播可能とする.

また，同じく動的伝播による変化は連続した値変化であることから，次の規則が成り立つ.

(h) 連続性規則

ある変数における動的伝播の実行に伴う静的伝播により，他の変数において不連続な変化が発生してはならない. 不連続な変化とは，変数の値が+から-，あるいは-から+へ，0を通過せずに変化するものである.

つまり，動的伝播に伴う静的伝播により，ある変数において不連続な変化が発生する場合は，その変数はそれ自身の動的伝播により連続的に0に変化すること可能なはずである. 不連続な変化とは，この0の状態を無視した変化であり明らかに不当である. de Kleer(3)の方法では動的伝播に伴う変数とその微分値の連続性に注目している. 本章で述べる方法では状態遷移は１つの動的伝播とそれに伴う静的伝播から発生するため，静的伝播の連続性にも注目する必要がある. ただし，打ち切り階数の値に対しては変化方向の同一性と同様に連続性は考慮しない. これは打ち切り階数の値に対しては高階の微分値からの動的伝播が定義できないためである.

この連続性規則からは，次の規則が導かれる．

(h-1) 連続性規則(2)

変数Aで発生した動的伝播に伴う静的伝播で，変数Bに不連続な変化が発生するとき，変数Bにおける動的伝播が変数Aのものより先行する．

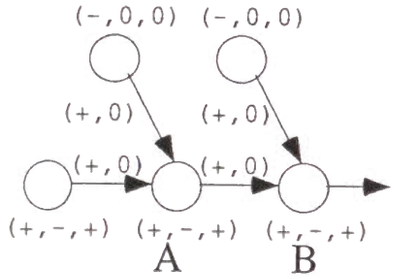

Fig.4.6 Example for continuous change

Fig.4.6の状態で変数A,Bの０階の値はどちらも０に変化可能である．ここで変数Aの値が０に変化したとし，この値を静的伝播させると、変数Cからの作用によりBの値は+から-に変化し，不連続な変化となる．このことは，Bの変化がAに先行すべきものであることを示している．このように，この規則は不連続な変化が発生した場合の，代替案となる処理を決定するのに用いられる．

また，一貫性，連続性などを調べる際に静的伝播を行った結果を計算しなければならない．この計算を前章で述べた静的伝播規則のみで行うと，その変数へ入力を持つ全ての変数を調べる必要があり，シミュレーション全体では大きな計算量になる．ここで，これまで述べてきた一貫性，連続性，０の瞬時性を考慮すると，値決定に用いることのできる次の２つの規則が導かれる．

(h-2) 静的伝播の値決定規則(1)（０からの変化）

ある変数で発生した変化が0から+あるいは0から-への変化であれば，変数値が0または?の変数のみ静的伝播により変化する．

(h-3) 静的伝播の値決定規則(2)（０への変化）

ある変数で発生した変化が+あるいは-から0への変化であれば，その値の静的伝播により発生する他の変数における定性値の変化は全て0への変化である．

つまり，(h-2)の０からの変化の場合，０の瞬時性から全ての０からの値変化は+,-から０への変化には先行するため、静的伝播による値変化も０からの変化のみが許される．(h-3)の０への変化の場合は逆に０からの変化は先行しているはずであり，同時に発生する伝播も０への伝播のみになる．

また，ある変数における動的伝播に対して，(g)(h)が満たされ静的伝播が可能な状態にあっても，静的伝播を行わなくても伝播の一貫性が満たされている場合は，その静的伝播は必要としない．このような場合はその変数における動的伝播がそのまま継続することになる．つまり，動的伝播と静的伝播の関係は次の３つが考えられる．

(1)動的伝播の結果が静的伝播可能であるが，必ずしも必要としない．
(2)動的伝播の結果が静的伝播可能で，かつ必要とする．
(3)動的伝播の結果は静的伝播不可能である．

実際の定性シミュレーションの際には，この３つの場合に応じて，処理を行う必要があり，その場合分けをこれまで述べてきた(f)-(h)の規則で行うことが可能である．

これまで述べてきた規則の全体的な関係を次に示す．

1. 動的伝播の実行
    (b)動的伝播規則
    (f)動的伝播の一貫性
    (d)動的伝播発生の決定則(*)
    (c)0の瞬時性(*)

2. 静的伝播の実行

(g-1)変化方向の同一性

(h-2),(h-3)静的伝播の値決定則(*)

(h-1)連続性規則(2)(*)

(f)静的伝播の一貫性

(a)静的伝播規則(*)

(d)動的伝播発生の決定則(*)

(e)動的バランス則(*)

ただし，(*)の規則から動的伝播が発生する．

ただし，(f)では注目している変数において一貫性が満たされない場合は，さらに静的伝播を実行することでその変数の一貫性が満たされるかを調べる必要がある．その他の場合は伝播が発生した変数とその周囲のみを調べることになり，比較的小さな計算量で済ませることができる．

４．４．２　変数の結合とブロック化

実際の定性ネットワークの作成には，各ユニット（パイプ，タンク等）ごとにモデルを作成し，それらの結合により全体を構築する枠組みを用意している．そのために本来ならば１つの値を示す変数が結合部において分割される場合があり，変数の一貫性を見直す必要がある．ここで、Fig.4.7のような場合を考えてみる．

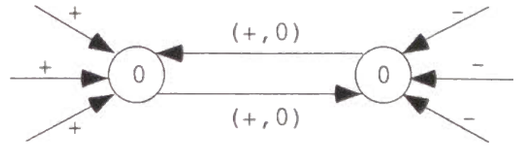

Fig.4.7 Variable connection

変数の一貫性のみを考えた場合，変数Aは+のみ，変数Bは-のみの入力であり，一貫性は満たさない．しかし，実際には変数A,Bは双方向アークにより結合されており，１つの変数が２つに分割されたものであり，A,B全体に対して+,-が入力されていると考えられる．前章で述べたオブジェクトの結合において、ユニット間の結合はこのような双方向アークを用いて変数を結合することで実現している．そのために，双方向アークを用いた結合に関しては，一貫性の定義を見直す必要がある．そのために変数ブロックという概念を導入する．次に変数ブロックの定義を示す．

Def.変数ブロック　互いに同じ作用を持つ双方向アークにより結合された変数の集合を変数ブロックとして定義する．

さらに，次の変数ブロックにおける伝播規則を加える．

(i)変数ブロックにおける伝播規則
変数ブロックによりブロック化されている変数に対しては，おのおのの変数を独立には扱わずに，変数ブロック全体ををを１つの変数とみて(a)-(h)の規則を適用する．

Fig.4.8において(1),(2)のように記述されたタンクとバルブのユニットを結合し，(3)のモデルが得られたとする．このとき(α),(β)で示される部分は変数ブロックとなる．このモデルは，これらを１つの変数とした(4)のモデルと同等に扱われる．

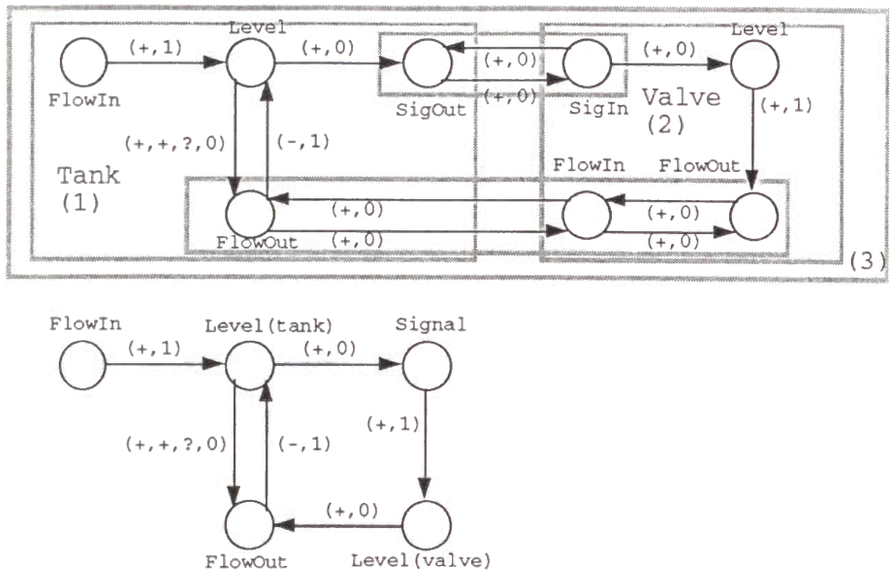

Fig.4.8 Example for variable blocks

変数ブロックの考え方を取り入れることで，ユニットの結合時にも一貫性の保たれたシミュレーションを可能とした．

4．5　他の定性シミュレーションアルゴリズムとの比較

この節では代表的な定性シミュレーションアルゴリズムであるKuipersの方法およびde Kleerの方法との相違点をまとめておく．

(1)deKleerの方法[5) 20)]との比較

deKleerの方法では、対象のモデル化に定性方程式を用いている。シミュレーションにおいては、状態間の時間的な変化を状態遷移として求め、その状態に対し定性方程式上で矛盾のない変数の組み合せである解釈から物理的に正当なものを選ぶことで、各々の変数の値を決定したり、物理的に起こり得ない状態遷移を省いたりする。そのため、

大規模なシステムを対象にすると、状態の数が膨大になり解析が困難になると考えられる。また、変数の数も多くなるため、各状態に対する解釈も増えてしまい、処理が複雑になる。本論文のアルゴリズムは、外乱をただ単に前向きに伝播させることでシミュレーションを行うことで処理を簡単化しようというものである。そのために、伝播を動的伝播と静的伝播に分け、動的伝播で状態遷移を発生させ、静的伝播でネットワークの一貫性を保証するという方法をとっている。この一貫性を調べるための規則は基本的にはdeKleerのものと同じであり、定性ネットワーク上のルールとして拡張されている。また、動的バランス則を用いることで前向きの伝播で発生する決定不能の変数を減少させている。

(2)Kuipers[4)]の方法との比較

Kuipersの方法では，定性方程式から得られるネットワークモデルを用い，ネットワーク上での値伝播と変数値の境界値への変化を基にシミュレーションを進める．この方法は，境界値に対する変数値の相対的な値とその変化方向に注目し，変数間に定義された作用に従ってそれらを伝播するというものである．この方法では、前向きの伝播を基にシミュレーションを進めるが，境界値の自動的な設定を行なうなど複雑な規則を用いている。本論文の方法はこの方法と同じく前向きに伝播を行うが，境界値が固定（0）で処理も簡単であり，(d)動的伝播発生の決定則を用いて独立な動的伝播しか調べないため，計算量が少なくてすむ．

4．6　シミュレーションの実行例

本節では，これまで述べた規則を用いて定性シミュレーションを行った例を示す．例としては，減衰振動を行うバネを用いる．その方程式は次の式で与えられる．

$$m \cdot d^2x/dt^2 + a \cdot dx/dt + b \cdot x = 0$$

ただし，定数m,a,bは全て正とする．これに対する定性ネットワークをFig.4.9(1)に示す．また，一貫性を分かりやすくするために各階数ごとに変数を分解したものをFig.4.9(2)に示す．この場合は，アークの作用は全て，-である．

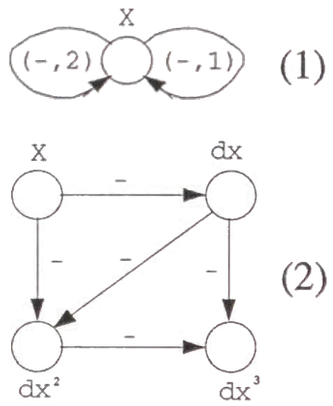

Fig.4.9 Example of Mass Spring System

ただし，計算には４階までの微分値を用い３階までの定性値を有効とする．初期状態としてx=(+,0,0,0)を与え，シミュレーションを行ったトレースをFig.4.10に示す．

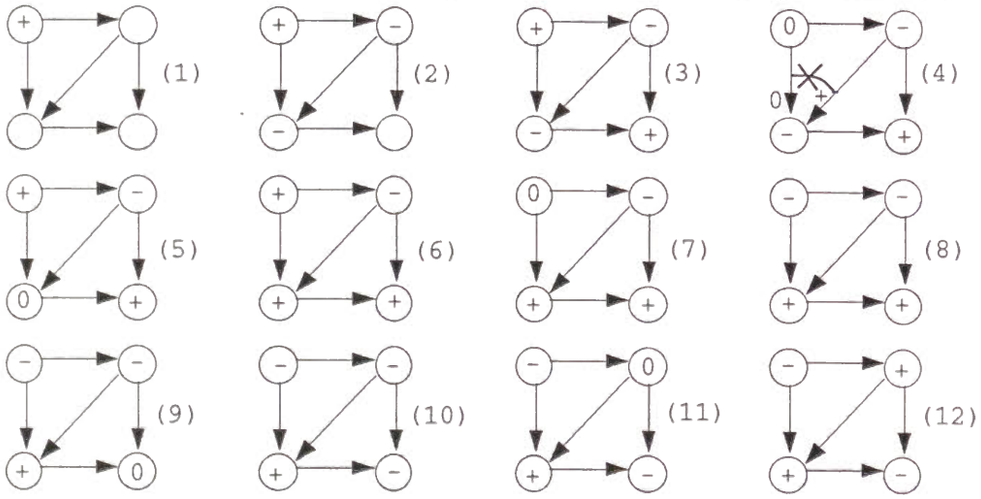

Fig.4.10 A sample of execution trace(1)

まず，第１ステップでの静的伝播により，Fig.4.10(3)の状態になる．第２ステップでは動的伝播が実行されるが，まず１階から０階への動的伝播を調べると実行後の結果は

Fig.4.10(4)になり２階の微分値の入力に矛盾が生じ，また実行結果の静的伝播も変化方向の不一致より無理で，実行不可能である．つぎに３階から２階への作用を調べると，Fig.4.10(5)になり，全ての入力で一貫性が満たされるため実行可能である．実行結果の静的伝播は変化方向が不一致より，実行されない．次には０の瞬時性からこの０が+に変化する．この場合は，一貫性は満たされており問題ない．さらに，この値は３階の値に静的伝播され，バランス則により３階の値に対し０への動的伝播がセットされる．その後は，Fig.4.10に示すように動的伝播が継続される．Fig.4.10の最終状態は(3)と比べると符号が全て逆であり，同様の伝播で(3)の状態に復帰し，振動状態が発生する．また，Fig.4.10の(6)の状態においては３階の値+も０から-に変化可能でこの変化により変数値は(+,-,+,-)となる。(Fig.4.11(1))この場合は，１階から０階の値への動的伝播が発生し，０階の値が０となり，この状態でネットワークの一貫性が満たされている。(Fig.4.11(2))

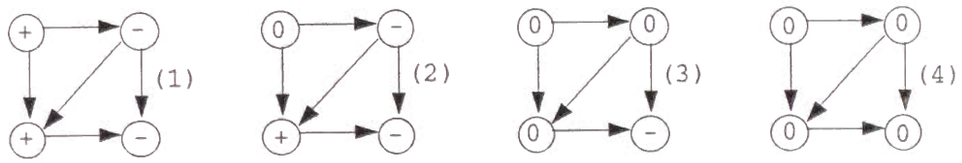

Fig.4.11 Sample of execution (2)

ここで，この値の静的伝播について調べると静的伝播は可能で，静的伝播を行うと，Fig.4.11(3)(4)になり，全ての値が０に収束する．これは，このすべての値が同時に０に変化することを表している．ただし，この変化は無限時間のステップを要する．

4．7　結言

本章では，定性ネットワークによる物理系の定性モデルの作成とその実行アルゴリズムについて述べた．次にその特徴を示す．

１.定性ネットワークによるモデル記述により，物理系の構造から容易に定性モデルを得ることができる．
２.新たに開発したネットワーク上での定性シミュレーションのアルゴリズムにより，高速に物理系全体の一貫性まで考えた定性シミュレーションが可能である．
３.対象が非線形の場合でも，線形要素と一部の非線形要素で記述可能な範囲でモデルを作成し，シミュレーションを行うことでが可能で，外乱に対する収束可能性等を調べることができる．

このアルゴリズムでは，動的伝播の実行の可能性とそれに伴う静的伝播の実行について判定を行うが，実際にはある時点において実行可能な伝播は数多く存在する．ネットワークの一貫性という意味では，これらの伝播は全て実行可能である．しかし，全ての場合に関して実行すると非常に多くのパスが存在するため，それらの中から実行する伝播を選択するメカニズムが必要になる．今回，作成したシステムでは変数，アーク等に優先度を定義し，優先度の高いものから実行させるようにした．この優先度はアークの結合される変数の係数の大きさなどを参考にして適当な値を設定しているが，実際には優先度をモデル作成時に決定するのは難しく，なんらかの方法で意味のある伝播を選択する方法が必要であると考えられる．

また，この方法ではネットワーク結合上での変数間の一貫性は調べているが，位相的な一貫性は調べていない．例えば２次の系で位相的な情報を扱うには，位相面上での変数値の挙動を考える必要がある．これまで位相曲線の交わりを防ぐなどの処理を考えた方法はあるが[24) 25)]，位相曲線の漸近的なふるまいなどを扱う方法は考えられていない．

この場合は，定性値をどのように位相面の情報として扱うかが問題になり，今後の研究の課題である．

第5章 結論

本論文では，定性的な手法によるダイナミックシステムのシミュレーションの方法について考察し，研究の過程で開発した２つのシステムについて述べた．これらのシステムが主な解析対象とするのは，大規模なために定量的な方法ではシミュレーションが不可能な系や，定量的なモデルは得ることが不可能で定性的なモデルしか作成できない系である．このような系のシミュレーションを行う際に問題となるのは，モデル作成の方法と，そのシミュレーションのアルゴリズムである．

モデル作成時には，対象が大規模であり，また類似した要素が複雑に結合された系であるためになんらかの系統的な手法が要求される．本論文では，この問題に対し，オブジェクト指向アプローチを適用した．

第２章では，プロダクションルールによるモデル記述に対して，オブジェクト指向アプローチを導入した記述言語を開発した．つまり，対象の各要素を，その要素が持つ属性と挙動を記述したルールからなるオブジェクトとして定義し，オブジェクト間の通信により要素間の作用を記述する．また，オブジェクトに対してはクラス，インスタンス間の継承関係により，多重継承を含む階層的なモデル記述を可能とした．さらに，オブジェクト間の通信リンクを導入し，リンクを用いたメッセージ通信により，オブジェクト間の構造的な関係の記述とメッセージによる作用の記述を分離し，物理的な構造をもつオブジェクトの記述を明解に行えるようにした．

第３章では，定性ネットワークモデルによる記述に対して，オブジェクト指向アプローチを導入した．定性ネットワークモデルは，対象に含まれる変数をノードとし，変数間の関係を属性を持ったアークで記述するモデルである．このモデルでは，対象の各要素を表すサブネットワークをオブジェクトとして定義し，それらを結合することで全体のネットワークを作成する．このシステムにおいては，多重継承の見直しとオブジェクトの結合によるモデル構築の２点で従来のオブジェクト指向アプローチを拡張した．多

重継承に関しては，従来の性質の継承である多重継承に加え，オブジェクトを組み合わせて新たなオブジェクトを定義するコンポーネント継承を導入した．コンポーネント継承はプラントのような物理的な構造を持つ対象をモデル化するのに有効な方法である．また，オブジェクトの結合時には，結合型を用いてオブジェクトの関係を記述することでオブジェクト内の変数間の結合に変換され，全体の定性ネットワークモデルが構成される．

この２つのシステムに共通した点としては，オブジェクトによる階層的な記述を可能にした知識記述とオブジェクトの結合によるオブジェクト間の関係の記述である．しかし，知識記述に関してはルールベースによるものとネットワークによるものという点で，オブジェクトの結合に関してはメッセージの伝達経路としてのリンクと変数間の結合としてのリンクという点で異なる．知識記述に関してこの２つの方法を比較すると，第２章のシステムの方がルールを用いた記述の自由度は高く汎用性もあるが，ルールにより物理作用の記述まで行う必要があり，人間に対する記述の負担が大きい，また，どちらかというとデバイスの状態間の作用を記述したマクロ的なモデルに向いている．それに対し，第３章のシステムは定性ネットワークに記述様式が固定されているため記述の自由度は低いが，物理作用に関する知識はシステムが持っているために記述の負担は小さく，どちらかといえば変数間の関係を記述したミクロなモデルに向いている．また，リンクに関しても同様に，第２章のシステムではメッセージ通信で処理するために，処理の内容をルールとして記述する必要はあるが，自由度は高い，逆に第３章のシステムでは変数の結合として定性ネットワークに展開されるため，特別な記述は必要としない．このように双方のシステムはそれぞれ特徴があり，その記述の様式および記述対象も異なってくる．いずれにせよ，このような枠組により，大規模な系に対しても階層的にモデルを作成することが可能で，また，以前のモデルの再利用やライブラリ化なども可能となる．

シミュレーションのアルゴリズムに関しては，大規模な系を対象とすることから，高速にシミュレーションが実行できるアルゴリズムが要求される．

第２章に述べたルールベースのシミュレーションシステムでは，一般のルールベースシステムでよく用いられるRETEアルゴリズムを，時間要素を条件部として持つルールの実行のために改良した時間RETEアルゴリズムを用いている．RETEアルゴリズムは，ルールの条件部をデータフローネットワークとして展開することでマッチングを高速化するものであるが，このアルゴリズムでは，RETEアルゴリズムでは直列に展開していた各属性に対する条件を並列に展開することで，属性に与えた条件にデーターフローネットワーク上の遅れ時間の形でそれぞれ独立に時間条件を与えられるようにした．また，オブジェクトの変更を表す命令であるmodify命令の処理の見直しにより，オブジェクトの変更時に、そのオブジェクトの持つ属性に対する条件の持続時間を正確に保持できるようにし，時間条件の処理を可能とした．さらに，オブジェクト間の継承や，リンクによるメッセージ通信の機能をネットワークに組み込むことで，これらの処理に伴うオーバーヘッドを軽減した．

第３章で述べた定性ネットワークモデルに対しては，第４章においてそのシミュレーションのアルゴリズムについて述べた．このアルゴリズムでは，定性ネットワーク上で前向きに変化を伝播することでシミュレーションを行う．ここで，この伝播を動的伝播と静的伝播に分けることにより，時間を伴う変化を１つの動的伝播を引金として発生させ，複数の変数で発生する同時変化は動的伝播の結果を静的伝播させることで行うものとした．このことで同時変化に対しては動的伝播の組み合わせを扱う必要がなく，処理が簡単になる．ここで，前向きの値伝播のみでは定性値が決定不能な変数が頻繁に発生するため，定性値の大きさを考えて平衡させる動的バランス則を導入し，決定不能な変数の発生を減少させた．また，ネットワーク上の変数値の一貫性や，発生した伝播の物理的な妥当性を調べながら伝播を行うためにいくつかの規則を導入し，物理的に発生可能な伝播のみを選んで発生させるようにした．さらに，第３章で述べたオブジェクトの結合時にも一貫性が満たされるように，結合に用いた変数を変数ブロックとして扱い，一貫性を調べるような規則を加えた．これらの規則は，定性ネットワーク上のシミュレーションと同時に実行され，計算機に対する負荷も小さくてすむ．

このように，本論文では定性シミュレーションを行うシステムに対して，そのモデルの記述とシミュレーションアルゴリズムについて考察を行った．また，実際にシステムを作成し，その有用性を確かめた．最後に，今後の課題としては次のようなものが考えられる．

１．モデル記述時にオブジェクトを用いたローカルな記述のみでなく，モデル化対象に対するグローバルな記述を可能とする．
２．定性ネットワーク上のシミュレーションに対し，より高度なヒューリスティクスを導入し，意味のあるシミュレーションのパスが選択されるようにする．
３．本論文で述べた２つのシステムの融合し，両方の特徴を生かしたシステムを開発する．

謝辞

本研究を行うに当り，懇切なるご指導と暖かい御援助をいただき，また本論文をまとめるに当り，論文の内容ならびに構成に関する御検討をいただきました京都大学工学部応用システム科学教室　得丸英勝教授ならびに石田好輝助手に心から感謝します．

また，本研究を遂行するに当って種々の御高配をいただいた徳島大学工学部知能情報工学科の皆様に厚く御礼申し上げます．

紀要論文

（1）時間制御機構を持つプロダクションシステム

－　RETEアルゴリズムの時間推論効率化のための改良

電子情報通信学会論文誌,pp1066-1076,Vol.J72-DII No.7,1989

（2）オブジェクト指向アプローチによる定性モデル構築

システム制御情報学会論文誌,pp287-297,Vol.3 No.9,1990

（3）定性ネットワーク上の定性シミュレーション

システム制御情報学会にて査読中

参考文献

1)石田好輝,:"知識ベース故障診断システムの動向",電子情報通信学会データ工学分科会DE87-12,pp1-8

2)T.H.Naylor:"ｺﾝﾋﾟｭｰﾀ ｼﾐｭﾚｰｼｮﾝ",培風館

3)淵　一博:"定性推論",共立出版

4)B.Kuipers:"Commonsence Reasoning about Causality:Deriving Behavior from Structure",Artificial Intelligence,vol24,1984

5)J.de Kleer,J.S.Brown,:"A Qualitaive Physics Based on Confluence",Artificial Intelligence,vol24,1984,pp7-83

6)J.D.Hollan,E.L.Hutchins and L.Weitzman:"STEAMER:An Interactive Inspectable Simulation-Based Training System",AI Magazine,Vol.5,No.2,pp15-27(1984)

7)K.D.Forbus:"Qualitative Process Theory",Artificial Intelligence,Vol.24,pp94-168,1984

8)竹内彰一:”オブジェクト指向の指向するもの”,情報処理,Vol.29,No.4(1988),pp295-302

9)G.M.Birtwistle, et al.:"SIMULA BEGIN",Studentlitteratur(1979)

10)D.Robson,A.Goldberg:"The Smalltalk-80 System",BYTE,Vol.8,No.8,36-48

11)L.Brown,et al:"Programing expert systems in　OPS-5",Addison-Wesley Publishing Company,1985

12)C.L.Forgy:"RETE: a fast algorithm for the many pattern/many object pattern match problem",Artificial Intelligence

13)Anoop Gupta and Charles I. Forgy:"Measurementson Production Systems",Technical Report of　　Carnegie-Melon University,CMU-CS-83-167,1983

14)M.I.Schor:"Advances in RETE Pattern Matching",Proc.AAAI-86 6th National Conf. on Artificial Intelligence,pp.226-231,1986

15)C.L.Forgy:"人工知能用言語　OPS83",パーソナルメディア

16)Y. Ishida and L.Eshelman:"Intergrating Model-based Diagnosis and Syndrome based Diagnosis -　A qualitative Approach to Process Diagnosis ", Technical Report of Carnegie-Mellon University, CMU-CS-87-111,1987

17)M.I.Schor:"Declarative Knowledge Programming: Better Than Procedual?",IEEE EXPERT SPRING,pp.36-43,1986

18)安西祐一郎:"階層構造を持つルールベース型表現を埋め込んだオブジェクト指向型知識表現言語OPHELLIAとその応用",コンピュータソフトウェア,vol.3,No.3(1986),pp43-60

19)Y.Ishida:"An Application of Qqualitative Reasoning to Process Diagnosis: Rule Generation by Qualitative Simulation",4th IEEEConf. on AI App.(1988),pp124-129

20)J.de Kleer,D.G.Boblow:"Qalitative Reasoning with Higer-Order Derivatives",Proceedings of AAAI-84,pp86-91

21)梅村恭司:"Lisp上のオブジェクト指向プログラミング",情報処理,Vol.29,No.4(1988),pp303-309

22)B.Falkenhaier,K.D.Forbus:"Setting up Large-Scale Qualitative Models",Proceedings of AAAI-88,pp301-306

23)R.Moore:"G2-A Real-time Process Control System",Electro/88 Conference Record,pp29/3/1-6

24)P.Struss,"Global Filters for Qualitative Behaviors",Proceedings of AAAI-88,pp275-276

25)W.W.Lee,B.J.Kuipers,"Non-Intersection Trajectories in Qualitative Phase Space: A Global Conetraint for Qualitative Simulation",Proceedings of AAAI-88,pp286-290